INFOPLEX DATA BASE COMPUTER

ARCHITECTURE -- CONCEPTS AND DIRECTIONS

Chat-Yu Lam

Stuart E. Madnick

August 25, 1978

CISR# 41

WP# 1046-79

This material is based upon work
supported, in part, by the
National Science Foundation
under Grant No. MCS77-2Ø829.

INFOPLEX DATA BASE COMPUTER

ARCHITECTURE -- CONCEPTS AND DIRECTIONS

Chat-Yu Lam

Stuart E. Madnick

August 25, 1978

CISR# 41

WP# 1046-79

This material is based upon work
supported, in part, by the
National Science Foundation
under Grant No. MCS77-2Ø829.

PREFACE

The Center for Information Systems Research (CISR) is a research center of the M.I.T. Sloan School of Management; it consists of a group of Management Information Systems specialist, including faculty members, full-time research staff, and student research assistants. The Center's general research thrust is to devise better means for designing, generating and maintaining application software, information systems and decision support systems.

Within the context of the research effort sponsored by the National Science Foundation under Grant No. MCS77-20829, CISR proposes to investigate the architecture of the INFOPLEX Data Base Computer which is particularly designed for large-scale information management. INFOPLEX applies the theory of hierarchical decomposition in its design and makes use of multiple microprocessors in its implementation to obtain high performance, high reliability, and large storage capacity. Research issues to be addressed include optimal decomposition of information management functions into a functional hierarchy to be implemented by a hierarchy of microprocessor complex, and optimal physical decomposition of an automatic memory hierarchy to support the memory requirements of the information management functions.

This report discusses the INFOPLEX concepts, functional and physical decomposition, and research directions.

ABSTRACT

The complexity, interdependencies, and rapidity of changes of events in our modern society have accelerated the need for more effective ways to store, process, and manage information. Conventional computers are primarily designed for computational purposes and are not well-suited for information management. This report introduces concepts and identifies research directions of a new computer architecture, called INFOPLEX, which is particularly suitable for large-scale information management.

The specific objectives of the INFOPLEX project include providing substantial information management performance improvements over conventional architectures (e.g., up to 1000-fold increase in throughput), supporting very large complex databases (e.g., over 100 billion bytes of structured data), and providing extremely high reliability.

By applying the theory of hierarchical decomposition, a highly parallel database computer architecture can be implemented by means of a multiple-microprocessor complex. In particular, INFOPLEX utilizes a memory hierarchy which can handle the storage and retrieval of a large volume of data efficiently. The information management functions are decomposed into a functional hierarchy to be implemented by

microprocessors. Decentralized control mechanisms are used to coordinate the activities of the memory hierarchy and the functional hierarchy.

The major research efforts of INFOPLEX include design, modeling, and evaluation of an optimal memory hierarchy, an optimal functional hierarchy and the associated distributed control mechanisms. Implementation of INFOPLEX using multiple microprocessors will also be investigated.

TABLE OF CONTENTS

1. INTRODUCTION

Confronted by an ever more complex society, it has become increasingly difficult to make decisions without sufficient information. Due to their enormous storage capacity and processing speed, computers have become a powerful tool for storing, processing, and retrieving information.

1.1 A Picture of the Future -- The Information Utility

In one picture of the future, we can foresee the evolution of an information utility (Madnick, 1977) where personal computers can be connected to information nodes (Figure 1). Even though each personal computer may have its own local database, a large shared database will still be needed for a variety of economic or technical reasons. Elementary versions of this system configuration already exist today in a variety of forms, including certain airline reservation systems and library information retrieval systems.

Another example of this system configuration is the New England Energy Management Information System (NEEMIS), which was developed jointly by M.I.T. and IBM (Donovan and Jacoby, 1975). In NEEMIS each user's modeling (or statistical) program is run on a different virtual machine (VM) and all

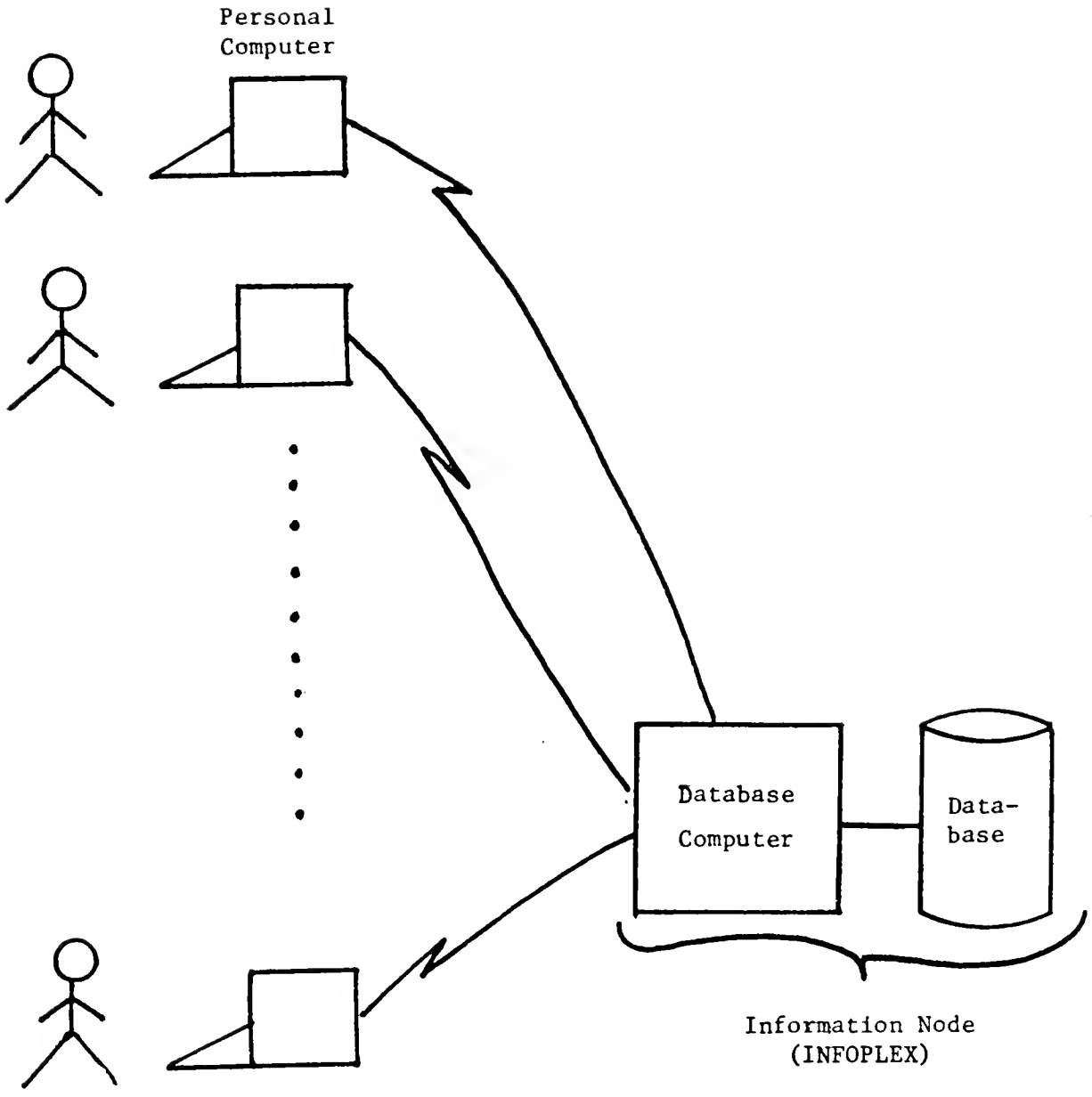

Figure 1. A Future Trend of Computing: The Information Utility

the modeling programs access a shared database through a database management system running on another VM (Figure 2). Conceptually, each VM can be viewed as a stand-alone computer. Therefore, we may view NEEMIS (Figure 2) as a software simulated version of the information utility depicted in Figure 1. Each modeling VM can be considered as a personal computer, and the database VM can be viewed as a database computer. NEEMIS has been effectively used for decision support by energy policy planners and researchers throughout the New England states. Therefore, the feasibility and usefulness of such an information utility is, to a certain extent, demonstrated by the NEEMIS system.

1.2 Desirable Capabilities of an Information Node

What capabilities should the information node in Figure 1 have in order to satisfy the demands of the users? We shall examine three important requirements: Performance, reliability, and size of the database.

Based upon our experiences with NEEMIS, in servicing a user's analysis requirements, the personal computer may need to generate many requests to the information node to retrieve or update data items. If the number of users is fairly large (e.g., 10,000) and the analysis extensive (e.g., 100 database queries per second generated by each personal computer), the information node could receive over

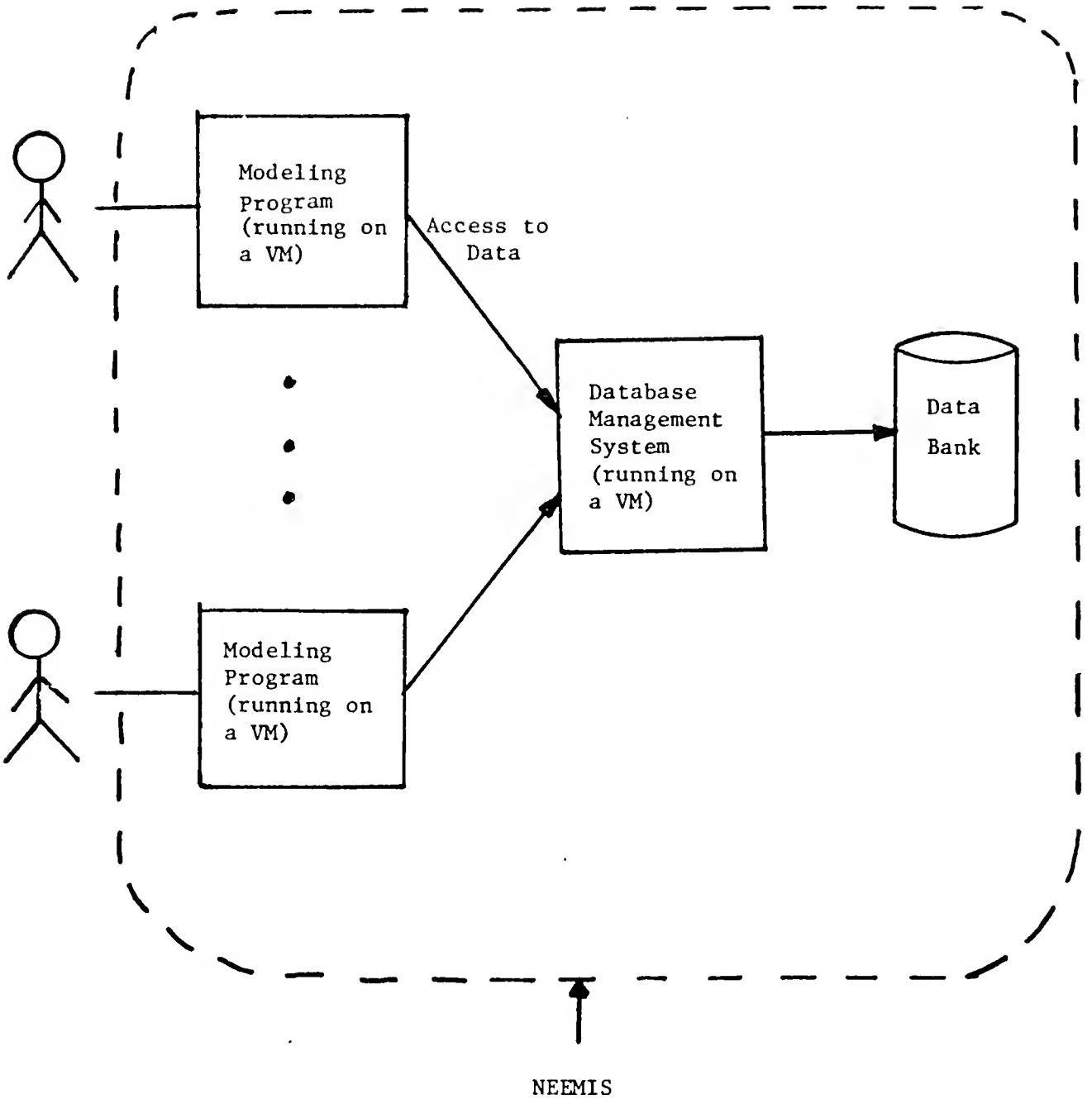

Figure 2. NEEMIS as a Software Version of the Information Utility

1,000,000 requests per second under peak load, well beyond the 10-100 requests per second capacity of most current day database systems. Therefore an information node must be capable of an orderly expansion from handling current day requirements to the requirements needed within the next few years (1000-10,000 requests per second), and to the expected requirements of the 1980's (up to 1,000,000 requests per second).

As noted above, typical current-day high performance database systems are capable of handling 10 to 100 requests per second, four to five orders of magnitude below the desired capability of an information node. Furthermore, although trends in computer hardware indicate continued decreases in cost, performance improvements in raw hardware speed are only expected to improve modestly. Thus, the performance improvement desired must be attained via novel highly parallel architectures.

Since the information nodes are critical to the operation of an information utility, reliability is a major concern. Due to the size and complexity of the software and hardware of conventional database management systems, it is difficult to attain such high reliability. A new approach, based upon more modular fault-tolerant hardware and simpler, less error-prone software, is needed.

The information node must also be able to store all the data needed by such a collection of users. Since there already exist databases today with more than 100 billion bytes of online data, it is reasonable to expect that there will be future requirements for an online storage capacity in excess of a trillion bytes.

Projections based on the cost/performance of current storage device technologies (e.g., IBM 3850 mass storage system), as well as more advanced experimental technologies, indicate that it will be economically feasible to attain the necessary storage capacities. But, no known existing computer architecture can support the necessary volume of database query requests nor provide sufficient reliability for such a large system. This paper will describe a research effort aimed at resolving this problem.

1.3 A Possible Solution -- INFOPLEX

The INFOPLEX database computer architecture proposed by Madnick (Madnick, 1975b) is a possible solution to the requirements for effective high performance, high reliability information nodes.

A key concept of the INFOPLEX architecture is hierarchical decomposition. The functions of information management are decomposed into a functional hierarchy,

referred to as the INFOPLEX functional decomposition. Each subfunction within a level of the hierarchy is implemented by means of a microprocessor complex. Highly parallel operations within each level and among levels is a major determinant of the high performance and reliability of INFOPLEX.

A large capacity, cost-effective memory with rapid access time is realized using a 'smart' memory hierarchy, referred to as the INFOPLEX physical decomposition. With a high degree of parallelism in operation, this 'smart' memory hierarchy is able to support the memory requirements of the INFOPLEX functional hierarchy. The control of the 'smart' memory hierarchy is distributed, and microprocessors are used to implement these control mechanisms.

By using both functional and physical decomposition, the INFOPLEX database computer architecture is able to realize a high degree of parallelism (thus high performance), modularity, and reliability. Modular implementation of the various functional and physical hierarchies is particularly desirable in order to respond to, and take advantage of, new information management techniques and memory technologies.

The INFOPLEX architecture is currently under study at the Center for Information Systems Research (CISR) in the M.I.T. Sloan School of Management. Other approaches to improve

information management capabilities are possible. We shall briefly discuss the merits and disadvantages of such approaches in the next section. In Section 3, we shall examine the INFOPLEX database computer architecture in more detail. Section 4 summarizes our directions in the INFOPLEX research.

2. RELATED WORK

The INFOPLEX is a new concept in system architecture for information management. In the past, computers were designed primarily for computational purposes. We now find that information processing has become a major, if not the dominant, component of computer usage. As Mueller (Mueller, 1976), President of System Development Corporation, noted: "The computer industry has gone through three generations of development to perfect machines optimized for 10 percent of the workload."

More recently, several ideas have been suggested for modifying the computer system architecture in order to handle information processing more efficiently. These ideas can be largely divided into the following four approaches: (1) new instructions through microprogramming, (2) intelligent controllers, (3) dedicated computers for database operations, (4) database computers. Most of the previous research activities in the field have focused on the first three categories. The INFOPLEX belongs to the fourth category. In the following sections, we shall discuss the advantages and disadvantages of these various approaches.

2.1 New Instructions through Microprogramming

The Central Processing Units (CPU) of most modern computers (as in Figure 3) use the approach of microprogramming in their design (Fuller et al., 1976). That is, the internal registers and functional units used to decode and execute the computer's instructions are controlled by a much more primitive microprogrammed control unit carrying out sequences defined in the high-speed microprogram memory (see Figure 3). For example, the microprogram determines how the 'multiply' operation is actually accomplished, typically by means of a number of shifts and additions. Thus, it is the contents of the microprogram memory that determines the computer's instructions as seen by a machine language programmer. This approach is often called microprogramming, microcoding, or firmware. By using a variety of microprograms, the same hardware may take on the appearance of a number of different computers; this is usually called emulation.

Conventional computer instructions are usually not well suited to the requirements of operating systems and database systems. Using firmware, it is possible to augment or enhance the instructions. Such an approach has been exploited extensively to support operating system functions in many contemporary computers, such as the IBM System/370 Models 138 and 148 (e.g., virtual machine assist).

Firmware can be used to enhance otherwise conventional

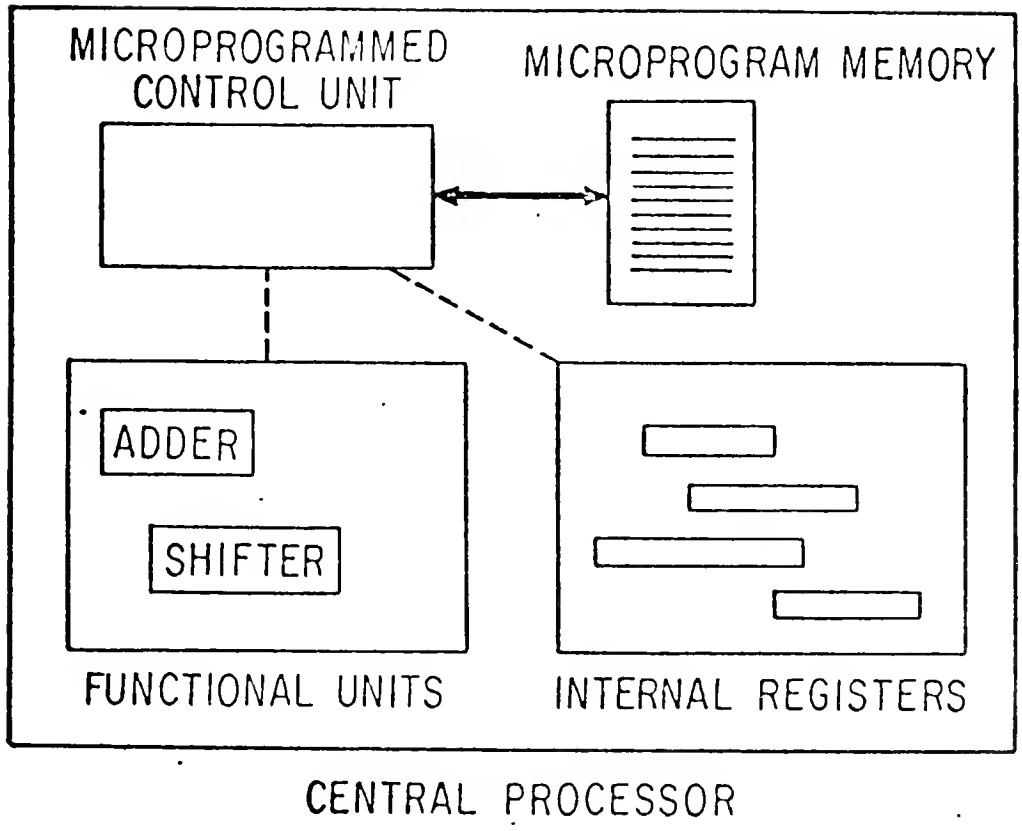

Figure 3. Microprogrammed CPU

computers to make them more suitable for supporting database processing. Instead of using inefficient subroutines, each database operation may be executed as a single microprogrammed instruction. This approach has been adopted in several systems. One of the earliest efforts occurred as part of the LISTAR information retrieval system developed at M.I.T.'s Lincoln Laboratory (Armenti et al., 1970), where several frequently used operations, such as a generalized List Search operation, were incorporated into the microcode of an IBM System/360 Model 67 computer. More recently the Honeywell H60/64 was announced with special instructions to perform data format conversion and hashing corresponding to frequently used subroutines of Honeywell's IDS database system (Bachman, 1975). The performance advantages of this approach are highly dependent upon the frequency of use of the new instructions and the extent to which they fit into the design of the overall database system software.

A more extreme approach is to define a completely new set of computer instructions, via firmware. In the past emulators have been developed that directly execute high-level languages, such as APL (Hassitt and Lyon, 1976). The Microdata REALITY System uses this approach to implement a complete database system in the microcode of an otherwise conventional minicomputer.

Since the basic hardware remains unchanged, the

performance improvement using firmware is largely gained by the higher speed of the microprogram memory (typically 2-10 times that of main memory) and the ability of microprograms to perform certain basic operations, such as bit manipulations, very efficiently. On the other hand, since microinstructions are usually quite primitive, it may take many microinstructions to perform a specific function. The overall central processor performance improvement using complete firmware emulation is typically in the range of 50% to 500% (Frankenberg, 1977; Hassitt and Lyon, 1976). Although such an improvement is attractive, it is not sufficient to close the gap (of four to five orders of magnitude) between current-day database system performance and the performance requirements of an information node as described in section 1.2.

2.2 Intelligent Controllers

Another approach to improving information processing efficiency is to use intelligent controllers. The controller provides an interface between the main memory and the devices (see Figure 4). Recently, more and more intelligence has been introduced into these controllers. For example, many controllers can perform the search key operation themselves (Ahern et al., 1972; Lang et al., 1977). As a matter of fact, some controllers are actually based on specialized microcomputers or minicomputers.

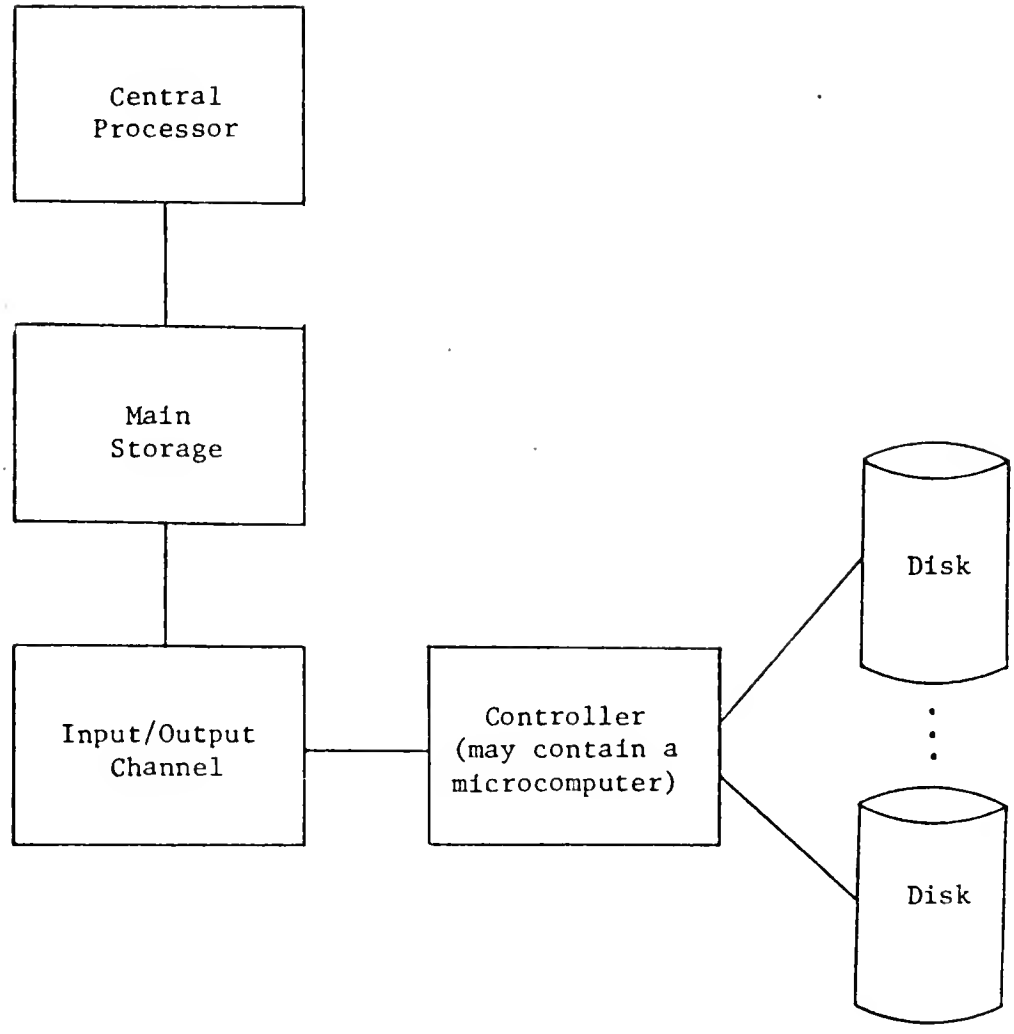

Figure 4. An Intelligent Controller

Two major types of intelligent controllers have emerged. The first type specializes in automating the data transfer between the storage devices, i.e., the physical storage management functions. The second type is designed to handle some of the logical storage management functions, such as searching for a specific data record based on a key. This latter type of device is sometimes referred to as a database computer. In the following sections we examine several examples of these two types of intelligent controllers.

2.2.1 Physical Storage Management

IBM's 3850 Mass Storage System as an example, uses an intelligent controller to automatically transfer data between high-capacity, slow-speed tape cartridges and medium-capacity, faster moving-head disks (see Figure 5a). From the point of the central processor, the 3850 appears as a large number of disk units (Figure 5b).

The 3850's real disks are used to hold active portions of the database, typically in units of cylinders (approximately 250,000 bytes). The controller maintains a record of the cylinders that are currently stored on the disks. When a request is received to read data from, say, virtual disk 2416 cylinder 36, the controller checks its storage location table to see if that cylinder is currently on one of the real disks; if it is, the requested data are immediately

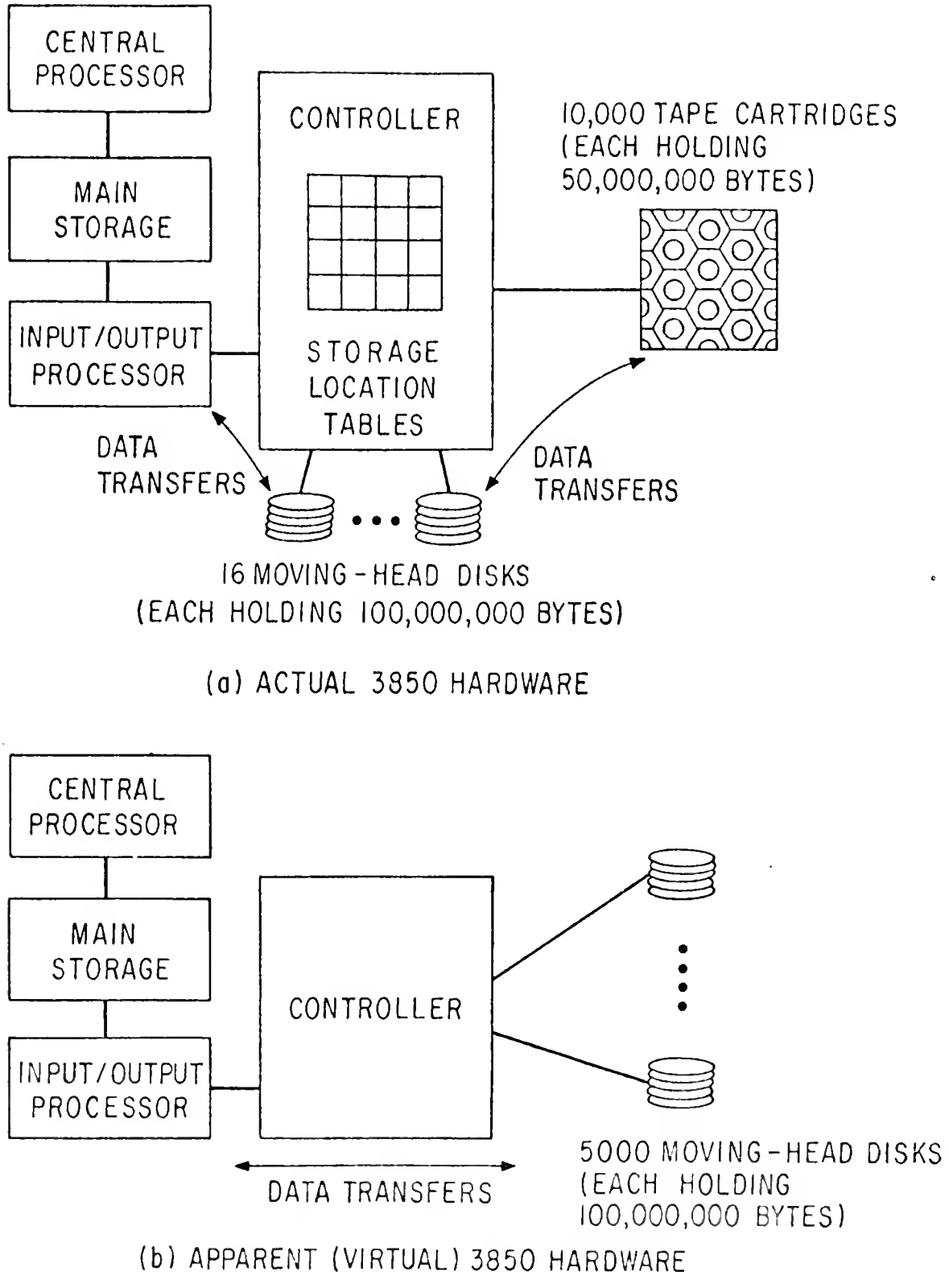

Figure 5. IBM 3850 Intelligent Storage Controller

transferred to main storage. If not, the corresponding tape cartridge (for example, cartridge 4868) is fetched and scanned to locate the correct cylinder, the entire cylinder of data is then transferred to an available cylinder on one of the real disks, and the storage location tables are modified correspondingly. The requested data are then transferred from the disk to main storage as described above.

When the requested data are already on the real disks, the request can be satisfied in about 25 msec; otherwise it may take 5 to 10 seconds. Except for this possible delay, the 3850 functionally performs exactly like 5000 disks but at only 2 to 3 percent of their cost. For applications where only small localized portions of a very large database are used at any time, the overall performance may be almost the same as having 5000 real disks.

2.2.2 Logical Storage Management

This second type of intelligent controller is used to perform associative or parallel searching (Langdon, 78). Most parallel associative search strategies are based on a head-per-track storage device technology (for example, magnetic drums, LSI shift registers, and magnetic bubbles) and a multitude of comparators, as depicted in Figure 6. As each data record rotates, either mechanically or

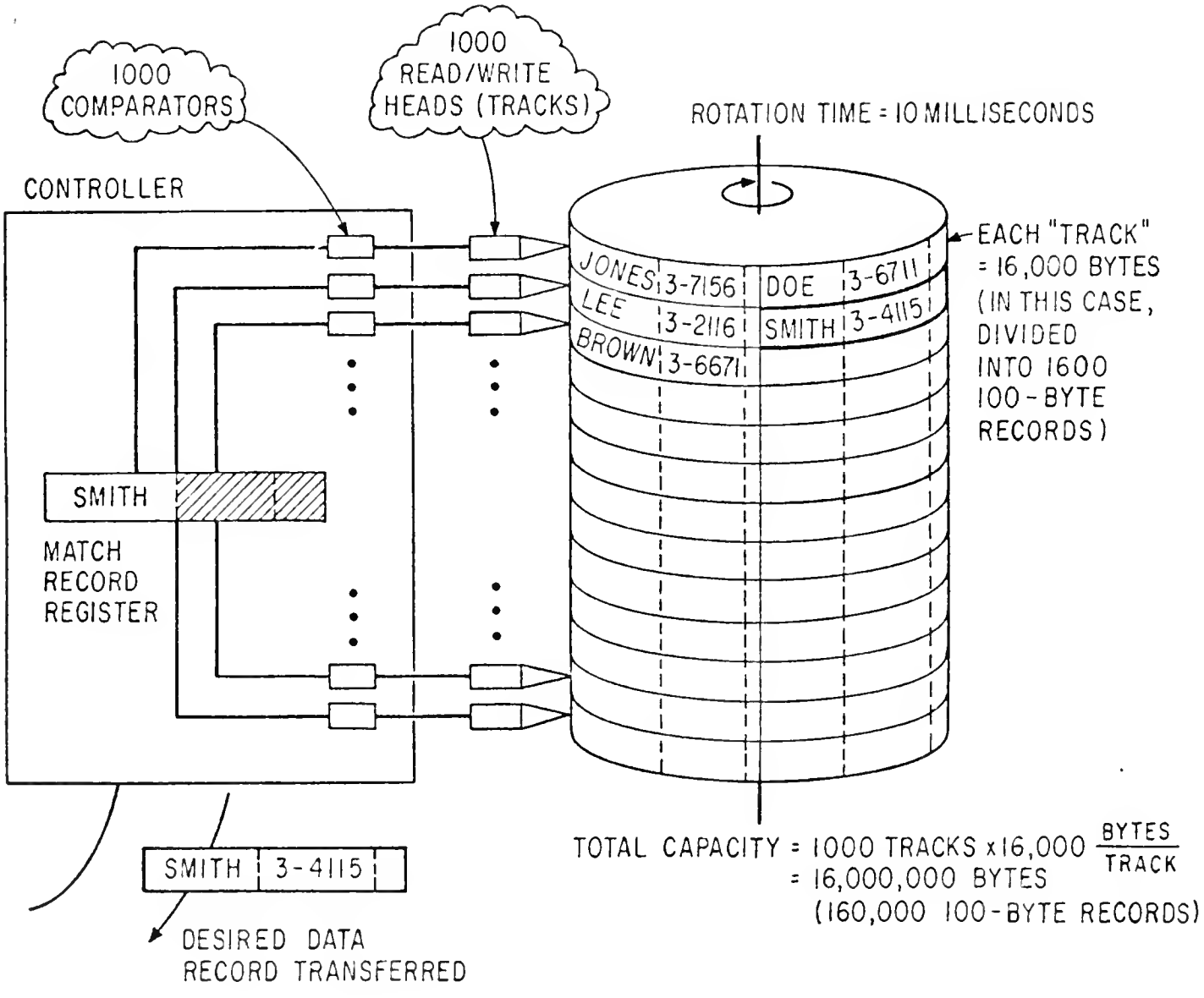

Figure 6. "Associative Search" Intelligent Controller

electronically, past a read/write head, it is compared with a match record register, called the mask. If there is one comparator per head per track, as in Figure 6, 1000 comparisons are done simultaneously; in this example a database of 160 thousand 100-byte records could be searched in about 10 msec, the rotation time for the storage device. In terms of conventional sequential search technologies, this is equivalent to reading and comparing one record every 60 nsec. In addition, the comparisons may be fairly complex, involving multiple data fields of the record.

The CASSM Project (Copland et al., 1973; Healy et al., 1972; Su and Lipovski, 1975; Su, 1977) was one of the first associative memory hardware projects to recognize the need for the data structures required for database systems and to design a device from the outset to support them. CASSM is specially designed to make use of sequential circulating memories such as disk tracks, CCDs, shift registers, or bubble memories. Data are organized as records which are packed into files. A file is then divided into equal size segments, each is then stored in a cell (e.g. a disk track). There is logic on each cell so that all the records at the same location on each cell can be simultaneously operated on.

The Rotating Associative Relational Storage (RARES) design (Lin et al., 1976) is aimed at providing a high

performance content-addressable memory for the realization of a relational database (Codd, 1970). The RARES hardware operates in conjunction with a query optimizer such as the SQUIRAL relational query language (Smith and Chang, 1975). Physically, RARES is connected to a CPU and buffer memory by a high speed channel. RARES uses a head-per-track rotating disk in which a relational tuple is stored orthogonally across several adjacent tracks (a band). A search module is associated with each band to perform access operations on the tuples in the band. The band organization greatly reduces the complexity of sending relational tuples to the CPU for processing. This is one example of how RARES was carefully laid out to facilitate the operation of other components. Another example of this is RARE's ability to maintain relational tuples in sort order or to rapidly sort tuples on a domain (i.e., on a record attribute) to facilitate certain kinds of search operations.

The Rotating Associative Processor (RAP) (Ozkarahan et al., 1975; Schuster et al., 1976; Ozkarahan et al., 1977) was also designed for a relational database. The basic RAP storage mechanism is a cell. Each cell consists of a microprocessor and a sequential rotating memory (e.g., a disk track, CCDs, or bubble memories). The tuples of a relation are stored as blocks of data on one or more cells. RAP has a high-level assembly language that is used to write RAP programs which execute relational queries. A RAP

instruction is executed by all the cells in parallel. A more recent RAP design (Schuster et al., 1976) also incorporates a virtual memory system for permanent storage of large amounts of data. Data are then "staged" from the virtual memory onto the RAP cells for parallel processing.

Although the decline in the costs of comparator electronics, due to advances in LSI technology, makes parallel search strategies quite promising for the future, they are only well suited to storage technologies that lend themselves to low cost read/write mechanisms, and for optimal performance and operation they tend to require a fairly simple and uniform database structure (e.g., relational flat files). To use these intelligent controllers in conjunction with other storage devices, such as mass storage, some "staging" mechanisms have to be used. Furthermore, these intelligent controllers only support part of the information management functions, much of the complex functions of language interpretation, support of multiple user interfaces, etc., of an information management system cannot easily be performed in these controllers. Thus, although these controllers may be well suited for certain specific information processing activities, their effectiveness in supporting an entire generalized information management system is likely to be more limited.

2.3 Back-end Processors

The third approach is to shift the entire database management function from the main computer to a dedicated computer (See Figure 7). Such a computer is often called a back-end processor.

The back-end processor is usually a minicomputer specifically programmed to perform all of the functions of the database management system. Although such a system could be operated independent of any other computer (stand alone), we will focus our attention on the cases where the back-end processor is connected to other processors or computers, either directly to physically close systems (tightly coupled) or through a communications system to physically remote systems (loosely coupled or distributed).

There are many advantages to this approach, including the following.

1. Low processor cost. Because of the processing characteristics of database management software, a low-cost high-performance minicomputer may perform as well as (or better than) a high-cost traditional processor optimized for mathematical calculations (for example, the ability to perform a 64-bit floating point multiply in 1 microsec is not generally needed in database processing.) By removing

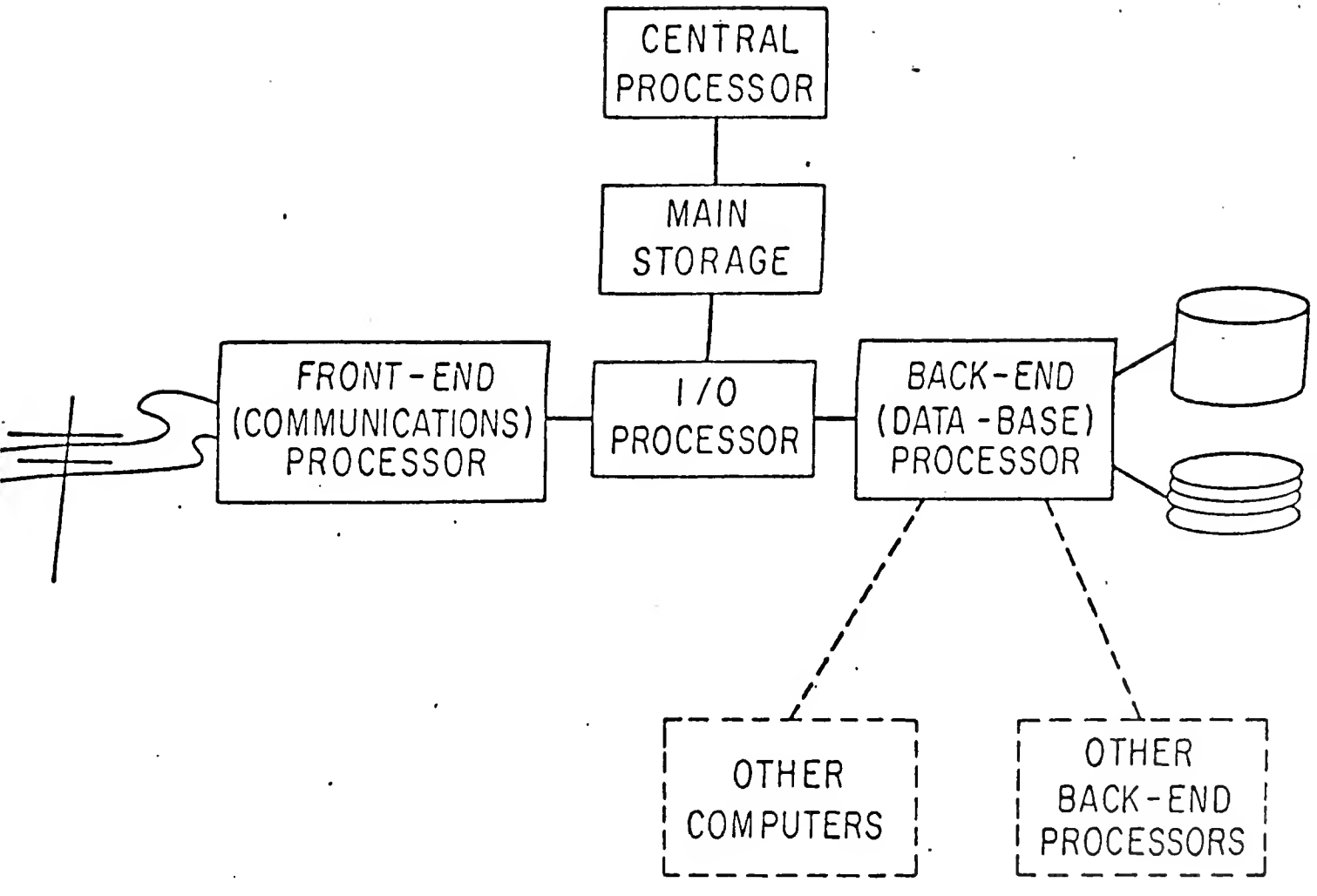

FIGURE 7.

BACK-END PROCESSORS

the database software load, the central processor can be more fruitfully used to service the remaining work load. The amount of load removable may be 40 to 50 percent or more (Heacox et al., 1975).

2. Sharing and distributed database processing. Back-end processors can serve as shared repositories of information accessible by all of the connected computers while at the same time enforcing defined information security constraints (Baum and Hsiao, 1976). The DATACOMPUTER (Marill and Stern, 1975) operating as a loosely coupled back-end processor through the ARPANET, an international communications network, can serve as the information repository for dozens of computers. In such an environment, multiple back-end processors, each serving as a database node, may be connected to provide distributed database processing capabilities.

3. Low storage cost. By pooling the storage requirements of many computers, more economical high-volume storage devices may be used. Many of the minicomputers and specialized processors on the ARPANET use the DATACOMPUTER because of its large capacity and low storage cost per byte.

4. Compatibility and uniformity. A large corporation or government agency may have dozens of different computers (often from different manufacturers). Interchange of

information between such computers has usually been awkward because of incompatibilities between the computers and lack of uniformity in the database. By use of the back-end processor concept, the information may be conveniently shared or interchanged among these disparate computers.

Back-end processors have evolved rapidly in recent years. Some of the earliest experimental efforts include the loosely coupled DATACOMPUTER (Marill and Stern, 1975), developed by the Computer Corporation of America using the DECSystem-10 computer, and the tightly coupled XDMS (Canady et al., 1974), developed by Bell Laboratories by modifying the firmware of a Digital Scientific META-4 minicomputer.

The commercialization of back-end database processors is proceeding down various paths, each offering different advantages to the user. As an example, Cullinane Corp., developers of the IDMS database system software for large-scale computers, has developed a back-end processor, based on the PDP-11/70 minicomputer, to be attached to IBM System/360 and System/370 computers. This facility may be very attractive to users of the older System/360. Relieving the 360, often a purchased rather than rented computer, of the database processing makes additional capacity available for the increasing application-dependent production workload. This can extend the useful lifetime of the installed computer and avoid a possibly costly conversion of

the production programs if it were necessary to upgrade to new computers.

In spite of all the advantages of the back-end processor, they cannot be expected to provide the significant performance improvements required of an information node discussed in section 1.2. The main reason is that the back-end processor is still a conventional computer whose architecture has been designed for computational purposes, not for information management.

2.4 Database Computer

The fourth approach to providing improved information processing efficiency is the database computer. The database computer has most of the advantages of the first three approaches. In many regards the database computer is logically the next step in the evolution of system architecture after new instructions, intelligent controllers, and dedicated computers.

The difference between this approach and the third approach (dedicated computer) is that the database computer has a system architecture particularly suitable for database operations while a dedicated computer merely adapts conventional computers to database applications.

There has been relatively little research on the development of true database computers (as opposed to work on intelligent controllers and/or dedicated back-end processors -- which are sometimes referred to as database computers). To the best of our knowledge, INFOPLEX is one of the few system architectures in the literature specifically designed to handle the very high database performance required by an information node. Other database computer research efforts include the DBC (Hsiao and Kannan, 1976) at the Ohio State University, and the GDS (Hakozaki et al., 1977) at the Nippon Electric Co., Japan.

3. INFOPLEX SYSTEM ARCHITECTURE

The INFOPLEX system architecture is based on the hierarchical decomposition of a large information management system. INFOPLEX uses both hierarchical functional decomposition and hierarchical physical decomposition. Figure 8 illustrates the INFOPLEX conceptual structure.

Each level of the functional hierarchy is implemented using primitives defined within the next lower level. Only the lowest level of the functional hierarchy interfaces with the physical memory hierarchy, which is regarded as a very large linear address space. The memory hierarchy (physical decomposition) is realized by a spectrum of storage devices and distributed control mechanisms that automatically manage the transfer of information among the levels of the memory hierarchy. By using multiple modules within a memory level and by pipelining requests between adjacent memory levels, a high degree of parallelism and reliability can be obtained.

The following two sections, 3.1 and 3.2, illustrate the functional decomposition and physical decomposition in some detail. Section 3.3 discusses implementation of the functional and physical hierarchies and the distributed control mechanisms employed. Section 3.4 summarizes the major advantages of the INFOPLEX architecture.

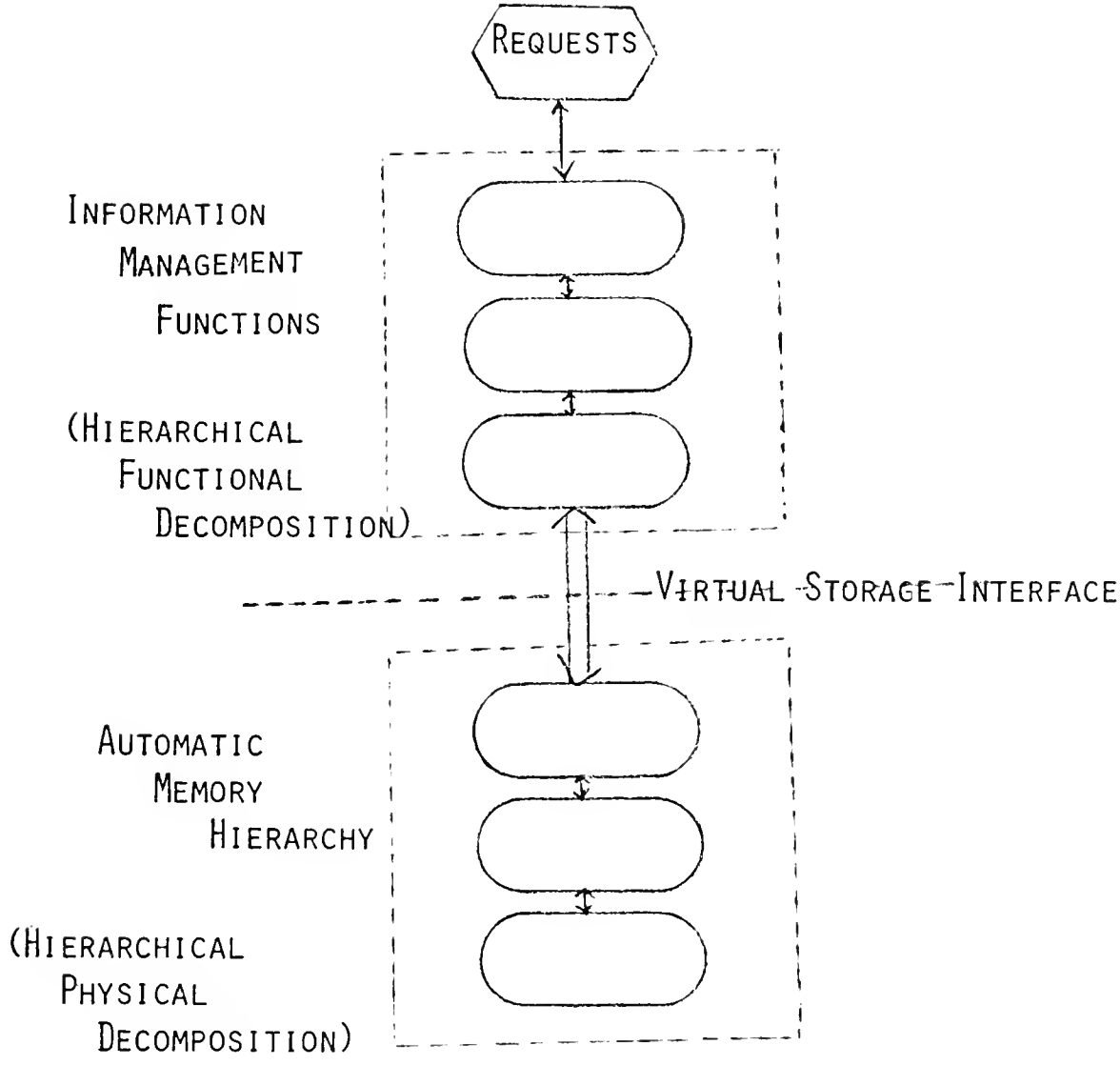

FIGURE 8 CONCEPTUAL STRUCTURE OF INFOPLEX

3.1 Functional Decomposition

An information management system performs a spectrum of very complex functions in response to user requests for information. These requests are often expressed using very high level languages and often come from many different sources simultaneously. There are many ways that these complex functions can be implemented but in our research (Donovan and Jacoby, 1975) we have found the technique of hierarchical functional decomposition to be very effective for advanced information systems. Similar techniques have been used successfully in operating systems (Dijkstra, 1968; Madnick and Donovan, 1974), basic file systems (Madnick, 1970), and a wide range of complex systems (Pattee, 1973; Mesarovic et al., 1970).

This is the approach we shall use in INFOPLEX. The information management functions are systematically decomposed into a functional hierarchy, referred to as the INFOPLEX functional decomposition. The functional modules in the hierarchy are then implemented using multiple microprocessors.

3.1.1 Rationale for functional decomposition

The central idea underlying the hierarchical functional decomposition approach involves decomposing the system into a hierarchy consisting of a number of levels, such that each level interacts only with the levels below it in the hierarchy. Proper selection of the hierarchy allows design or operating problems that previously impacted the entire system, to be isolated to one or a few specific hierarchical levels, and thereby more easily handled (Parnas, 1975).

Isolating the information management functions into minimally interrelated modules facilitates the use of multiple identical modules for performing the same function, so that reliability and parallelism are enhanced. Furthermore, this approach provides great flexibility in the technologies used for implementating each type of functional module. For example, a particular data structure may be selected from a spectrum of indexing techniques for a given module without affecting the design of other types of modules.

3.1.2 Example of a functional decomposition

To illustrate the hierarchical functional decomposition concept, we shall discuss a specific example of a functional decomposition in this section. Figure 9 illustrates a

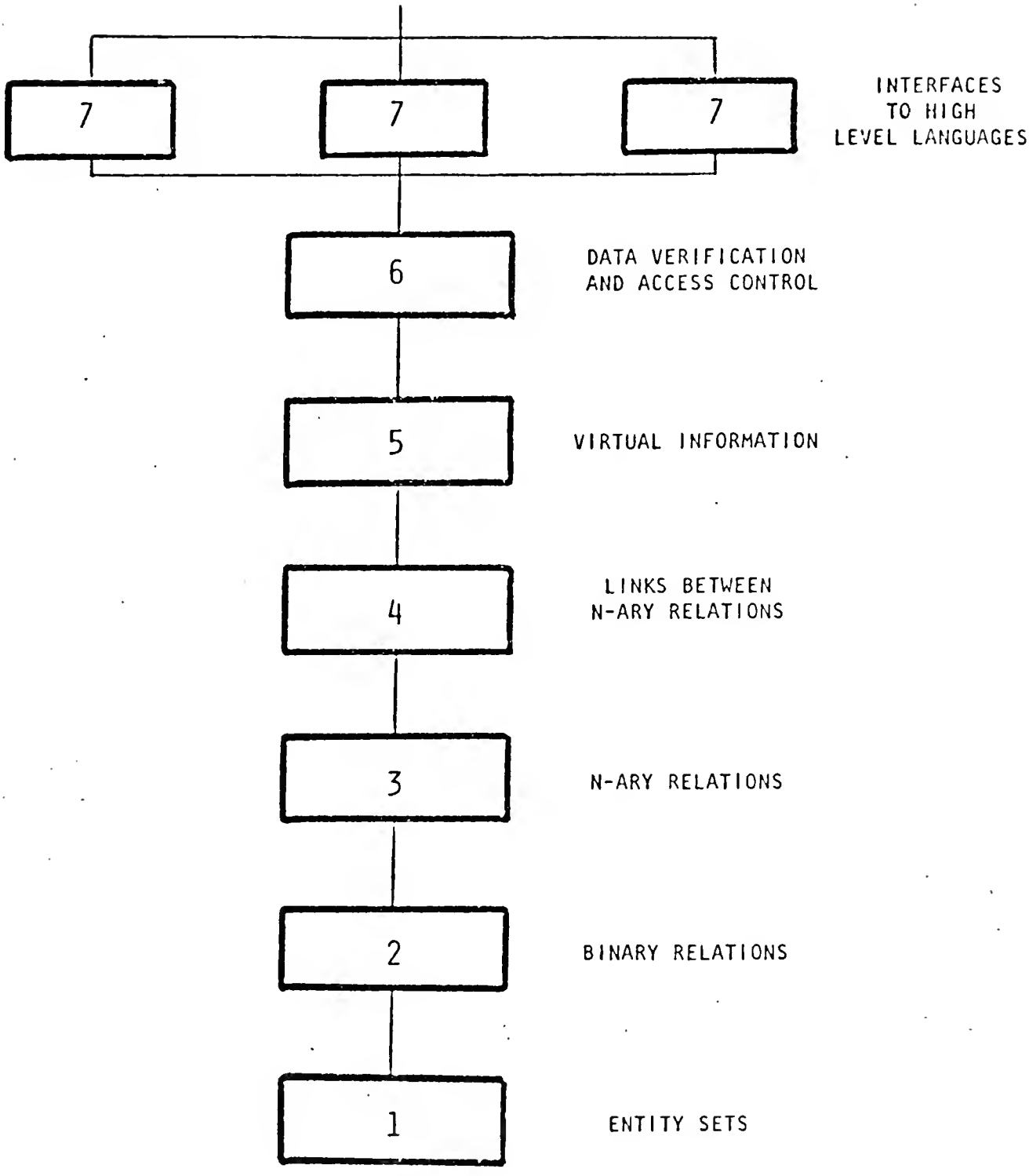

Figure 9. Example Functional Decomposition

plausible hierarchical functional decomposition. Each level of the functional hierarchy is described below.

3.1.2.1 Entities and entity sets

At the most fundamental level, a database system stores information about things, or entities. Also, it is usually the case that entities represented in a database fall naturally into logical groups, or "entity sets". The way in which a database system (a) represents and stores information about entities themselves, and (b) represents information about the logical grouping of entities into entity sets, forms the bedrock architecture of the system.

There are many schemes available for logically and physically representing entities (i.e., coding, storing, and addressing entities) and various algorithms for structuring entity sets. The choice of implementation scheme at this level affects the performance of the entire system but does not affect how the functions of the other levels are implemented.

3.1.2.2 Binary relations

All relationships among entities can be expressed in terms of binary relationships between pairs of entities. This functional level makes use of the entity level

constructs to provide a collection of binary relations (relations between pairs of entity sets). An element of a binary relation can be viewed as a triad, consisting of a relation identifier plus two entities, each from one of the entity sets participating in the binary relation. Thus a binary relation can be viewed as a collection of triads with the same relation identifier.

Perhaps the simplest possible implementation of a set of binary relations would be as a sequential file of triads, for example,

```
(HAS_SALARY_OF ,  SMITH ,  1200)
(HAS_SALARY_OF ,  JONES ,  1500)
        ...
(WORKS_IN_DEPT ,  SMITH ,  02)
(WORKS_IN_DEPT ,  JONES ,  07)
        ...
```

The difficulties with this approach are manifest: there is great data redundancy and thus waste of storage (the relation identifiers are stored in each triad); insertion of additional triads would either have to be done out of order, or else insertions and deletions would be extremely time-consuming.

Triads could also be stored as linked lists.

Alternatively hashing algorithms could be employed to locate any triad, given two of its three components. The use of linked lists can improve access speed and reduce storage requirements. On the other hand, the use of hashing algorithms would provide extremely rapid access, but would be poorer in terms of storage space utilization.

Since a database may contain billions of triads, the logical and physical structures of binary relations have serious performance implications. Many implementation schemes for binary relations are possible. Although the choice of these implementation schemes has various cost and performance implications it does not affect how the functions of the next level are implemented.

3.1.2.3 N-ary relations

Conceptually, an n-ary relation may be thought of as a table of data, with rows of the table (usually called tuples) corresponding approximately to records in a traditional data file, and columns (or domains) corresponding to fields. Furthermore, n-ary relations may be constructed out of sets of basic binary relations. For example, the degree 4 relation EMPLOYEE_DEPT_SALARY_SEX, for which a typical entry might be

(SMITH, 02, 1200, male),

is semantically equivalent to (i.e., contains the same information as) the three binary relations WORKS_IN_DEPT, HAS_SALARY_OF and SEX, as illustrated in Figure 10.

We could build up n-ary relation tuples out of tuple-ids of binary relations, as illustrated below, in Figure 11. In this approach, the original data entities (SMITH, 01, 1200, male), would be stored in permanent binary relations, and all other relations would be constructed out of binary tuple ids. Tuple ids, being uniform binary numbers, are easy and efficient to manipulate.

A number of other implementations of n-ary relations is also possible. The point is, however, that once we have an efficient implementation of binary relations, general n-ary relations may be constructed in a straightforward fashion out of the binary relations without actually having to retreat -- conceptually or physically -- back to the level of basic entities or entity sets. In other words, n-ary relation functions (to manipulate n-ary relations) can be implemented by appropriately combining binary relation functions.

3.1.2.4 Links among n-ary relations

The various n-ary relations in a typical database would

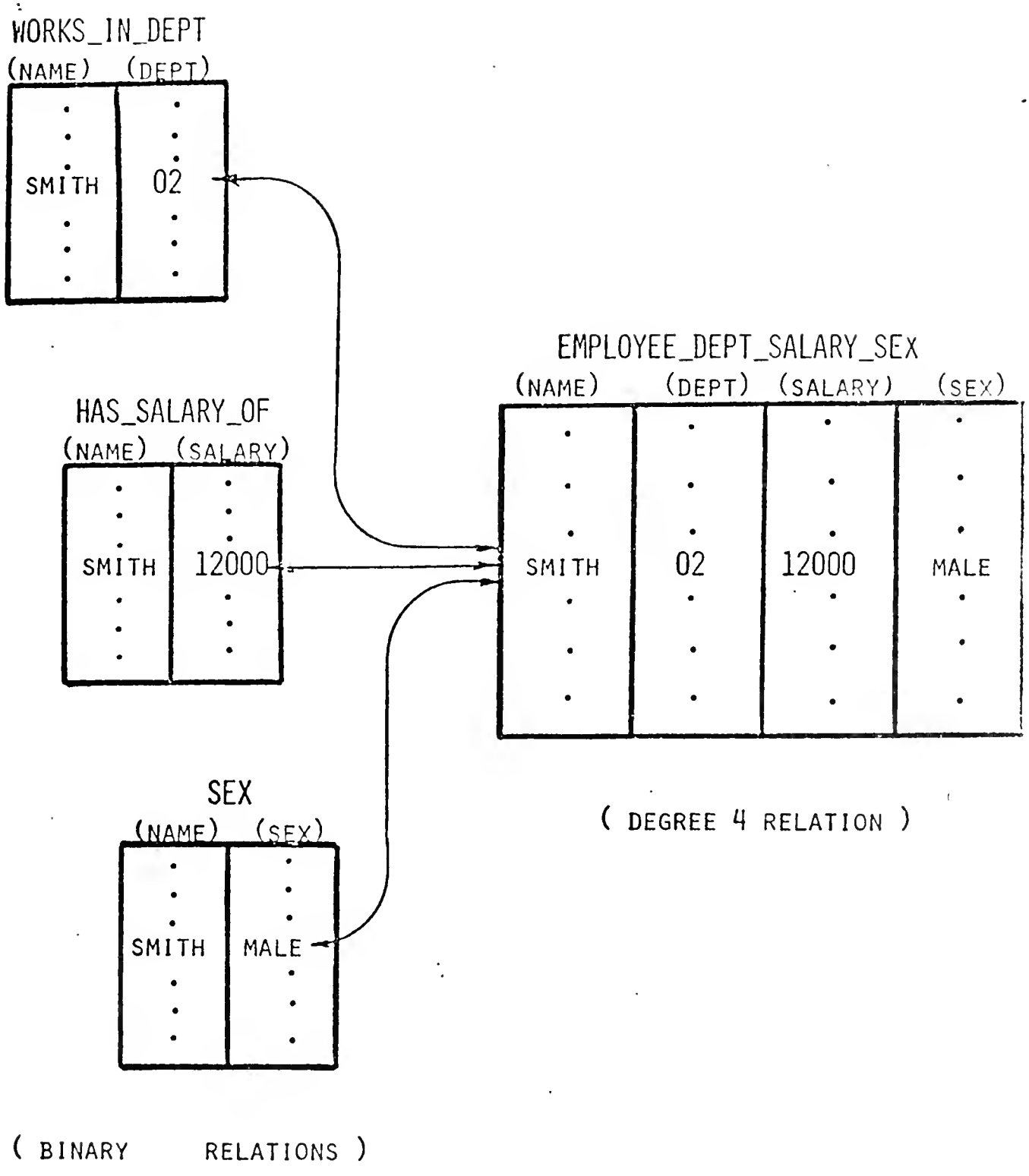

Figure 10. Construction of N-ary relation from Binary relations

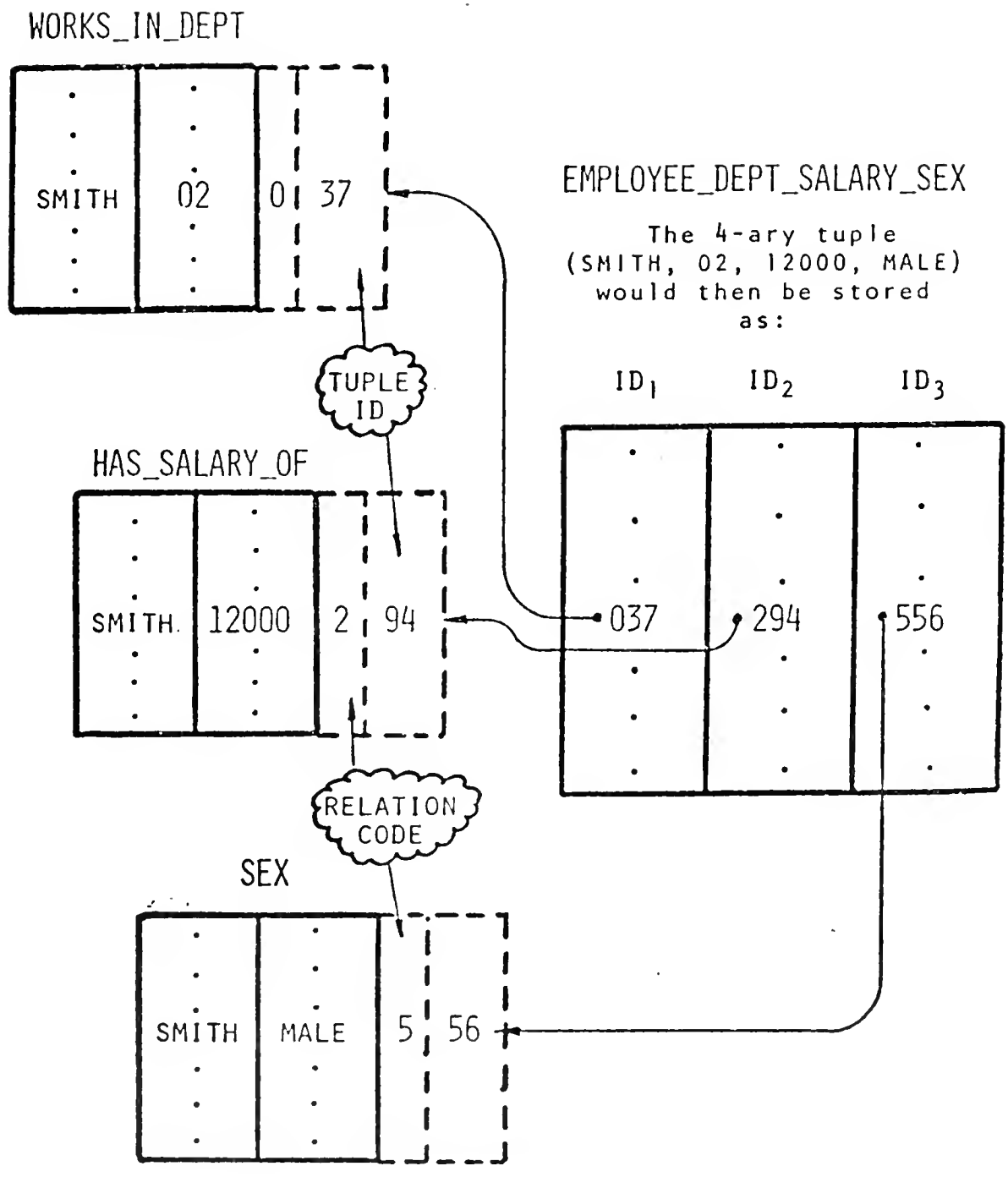

Figure 11. Use of tuple-ids in N-ary relation

generally possess a number of logical interconnections. For example, one relation might contain data on employees and the skills each employee possesses, while another might involve data on departments and the skills each department requires to function. The logical relationship between the tuples in these relations could be employed to extend the database structure further, by incorporating a set of "meta-relations" for storing information about such links between the regular n-ary relations. The role of the meta-relations would be to identify related tuples, and to provide some semantic information regarding the nature of the interrelationships. In the example cited above, it would make sense to establish a meta-relation connecting the appropriate tuples in the original two relations on the basis of "common skill", as shown in Figure 12.

Under the implementation approach illustrated in Figure 12, meta-relations would themselves be n-ary relations. The only difference between them and regular n-ary relations lies in the interpretation of their entries. Therefore, all of the previously designed mechanisms for building and managing n-ary relations could also be used with the meta-relations. Only the interpretation of the elements within these relations would be different.

By incorporating linking information among the different n-ary relations in a database, either permanently or

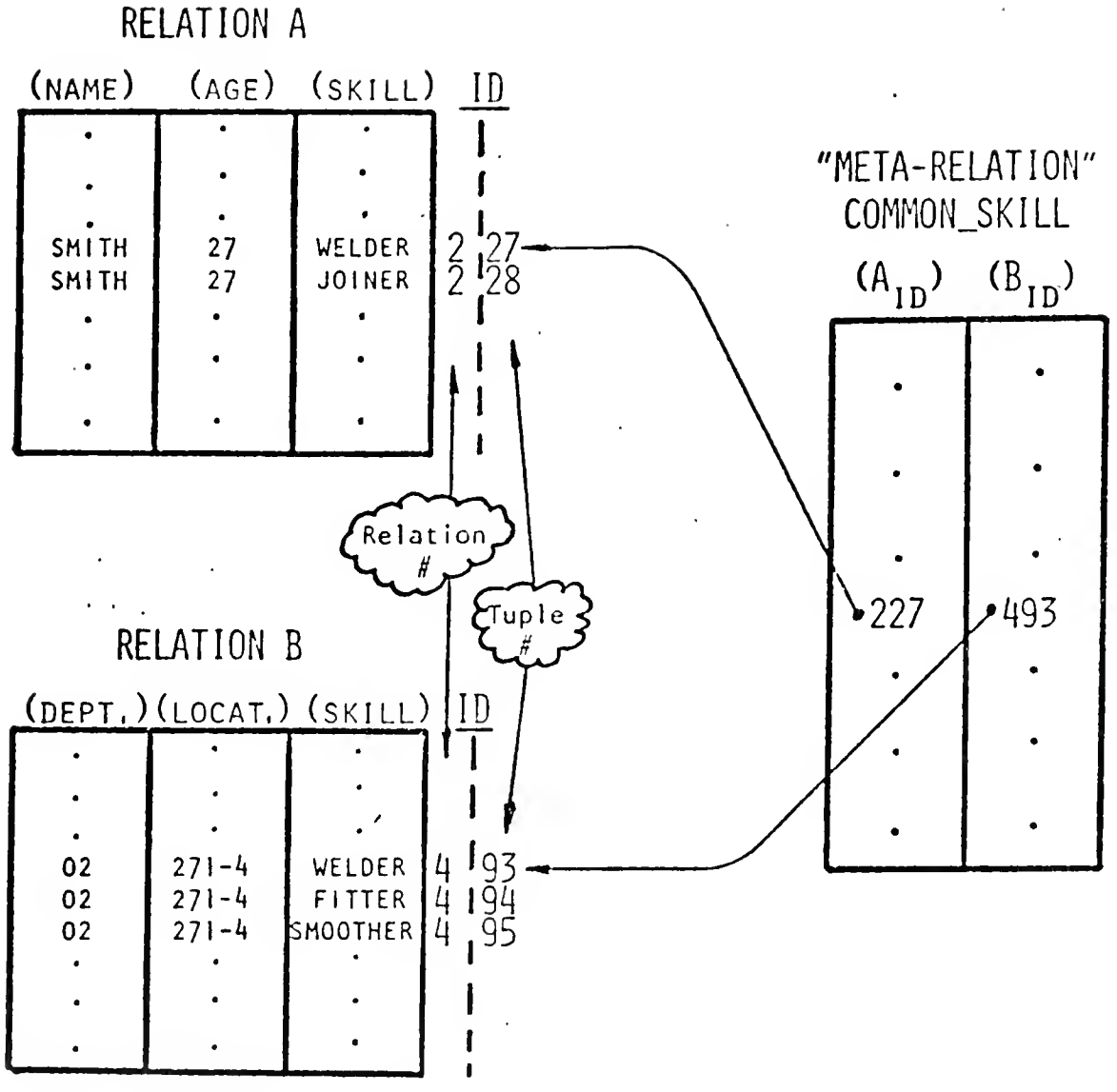

Figure 12. Linkage between N-ary relations

temporarily, directly into the database structure itself, it would be possible to generate more complex systems that would be capable of presenting different interfaces to different users, depending on the needs and objectives of the users themselves.

3.1.2.5 Virtual information

It is not always necessary, or even desirable, that a database contain all the information that users might wish to access. Sometimes data interrelationships are algorithmic in nature, such that certain values may be unambiguously derived from others that are already stored in the database. This gives rise to the concept of "virtual" information (Folinus et al., 1974).

If an employee's BIRTH_DATE is stored in a database, and the CURRENT_DATE is also available, then the employee's AGE could be calculated by a simple algorithm and need not also be stored. If this is in fact done, then the employee's AGE would be an example of "virtual" data -- information that appears (to the database user) to be stored there, but which is not actually present as an entity in the database.

There are a number of advantages to "virtualizing" data in a database. These include:

- greater accuracy: for example, an employee's AGE could be calculated as accurately as necessary if included as virtual data, whereas it would always be somewhat "old" if it were simply stored as a database entity;

- elimination of updating: virtual data items themselves never need updating;

- reduced redundancy: including, for example, BIRTH_DATE, CURRENT_DATE, and AGE as three separate items in a database is redundant, and inconsistent data relationships can easily result if some of the items are updated independently of others;

- savings in storage: in many cases, the database storage space required to store items such as AGE directly would be much larger than that required to store the coded algorithm for calculating AGE from other data.

One way of implementing a virtual information capability is to extend the definition of n-ary relations to include tuple identifiers ("ids") that would in fact not refer to binary relation tuples, but rather would point to procedures for calculating the virtual data items. Consider a simple employee relation of degree four, containing real data items

NAME, BIRTH_DATE, and SALARY, plus a virtual data item AGE. The organization of this 4-tuple would then appear as in Figure 13.

3.1.2.6 Data verification and access control

Data verification is the process of checking entries into a database for qualities such as reasonableness (e.g., a person's age should be no greater than, say, 125 years), and consistency (e.g., the sum of the months worked in various departments by an employee should sum to the number of months worked for the company). Access control is the process of controlling the database with regard to data retrieval, update, deletion, database reorganization, etc. For example, department managers may be granted authorization to view the employee records of only the employees working in their own departments; the database administrator, on the other hand, may have access to all the records in the database. The database administrator may also be the only person with authority to reorganize the entire database.

Access control also involves considerations such as the identification of valid users through use of passwords and other such techniques, mechanisms for allowing users to specify the type of access (read only, read/write, execute only, etc.) for files, and allowing users to segment files,

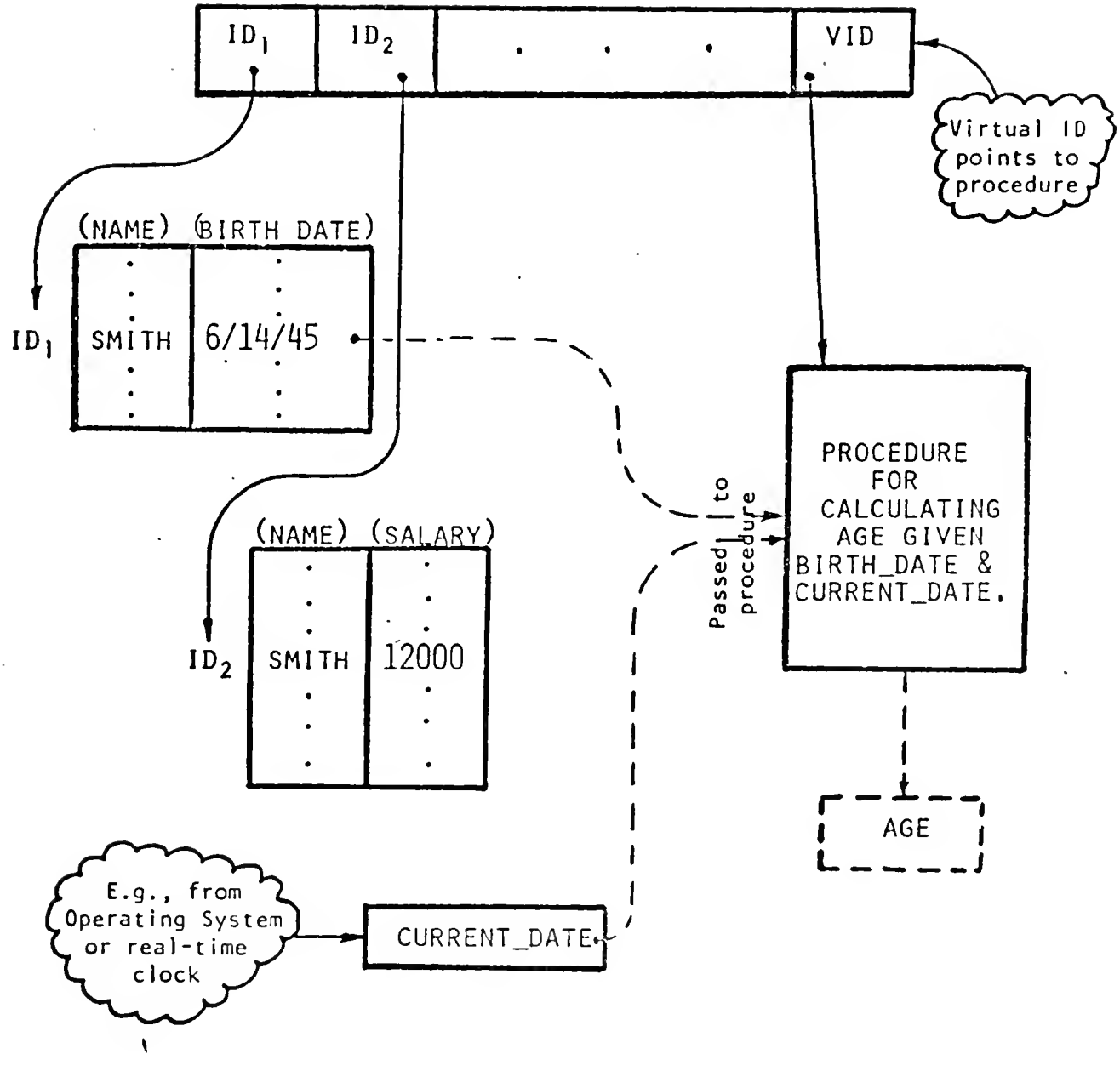

Figure 13. Virtual Information

so as to restrict parts of interconnected programs or data files from certain kinds of access by certain specified users (an example of a system that has implemented this successfully is the MULTICS system).

Both data validity and access control could be implemented in the hierarchical structure being discussed here in a variety of ways. For example, the basic n-ary relations could be further extended to include special control and verification tuples. If data verification were to be performed upon data entries in a certain domain of a relation, that domain could be flagged in a "verification tuple", and a data verification routine would be called upon data insertion or update to check the appropriateness of each entry (see Figure 14).

Similarly, control of access to various domains or tuples could be performed by setting control bits in a special control tuple or domain, and including, for example, an address pointer to a list of authorized user passwords, against which the current user could be checked. These control tuples or flag bits would serve to describe certain "views", or combinations of data elements, that each user would be permitted to access. Alternately, they could be used to describe elements, domains, tuples, or entire relations that a user was not permitted to view.

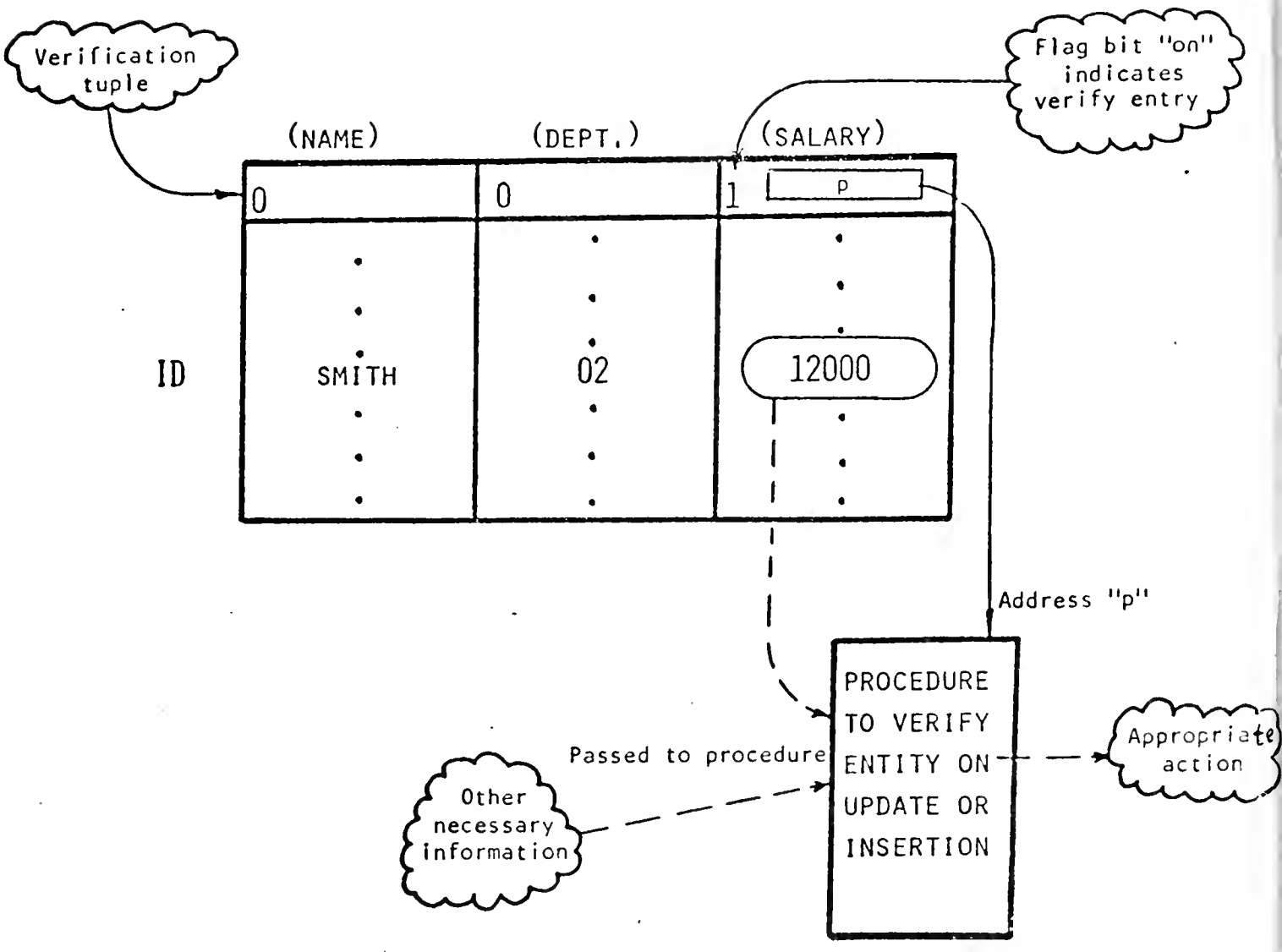

Figure 14. Verification and Access control

Note that these implementations would utilize the mechanisms employed to provide virtual information as discussed above (i.e., certain ids are used to point to verification procedures, as they pointed to "virtual information computation procedures" in the preceding section). Thus, the verification and access control functions can be realized in terms of those responsible for virtual information.

3.1.2.7 High-level language interface

The user interface, through the data manipulation language, basically specifies the way in which the database may be accessed by the users. In this regard, there are three main approaches to manipulating a database, corresponding roughly to the three basic models of database organization (network, hierarchical, and relational.):

1. An applications programmer may wish to "navigate" (Bachman 1973; Codasyl 1971) a database by using the data manipulation language to trace through the data groupings (relations) and interconnecting linkages (links between n-ary relations). This approach to database manipulation is usually more complex than some others, and demands a greater sophistication on the part of the applications programmer. He must, for example, be fully aware of the existence of all the

links connecting the various data groupings, whereas this knowledge is not necessarily demanded of programmers using other data manipulation languages. In return for the greater complexity, the navigational approach usually offers greater accessing efficiency and better overall database manipulation performance, especially when dealing with large and complex databases.

2. A user may wish to organize and manipulate the database as a hierarchical tree structure, wherein the logical interconnections between data groupings are always one-to-many in nature. In a sense, the manipulation of a hierarchical tree structure is a special case of the general navigational approach. Hierarchical structures do, however, allow a number of simplifications to be made in designing the database management system, as well as in the data manipulation language. Furthermore, a surprisingly large number of situations in the real world may be effectively represented with a hierarchical tree data organization, so it is worthwhile to treat hierarchical structure as an important special case.

3. Finally, in many cases it is appropriate for the applications programmer to access the database directly in terms of its underlying binary or n-ary relations

(Codd, 1970; Codd, 1974). Such "direct" manipulation may be made at a relatively low level, in terms of individual relations and primitive operations (using the relational algebra) upon them. Alternately, a higher-level interface could be used to translate more general-purpose commands (using the relational calculus) into lower-level operations. Such low-level accessing methods generally provide greater efficiency, at the expense of greater programming detail.

3.1.3 INFOPLEX's approach to functional decomposition

The above discussions illustrate one possible decomposition of the information management functions into hierarchical levels. Other decompositions are possible. For example, the work of (Senko, 1975; Yeh et al., 1977; Toh et al., 1977; ANSI/SPARC, 1975) also decomposes the various information management functions into several levels (e.g., (1) physical data storage, (2) logical data encoding, (3) access path, (4) internal schema, and (5) external schema). A common weakness of these functional decompositions, including our example decompositon, is that although any particular decomposition may make good sense and impose a reasonable conceptual structure on the information management function, there are no commonly accepted criteria with which to evaluate any given decomposition.

A common qualitative criteria often used to decompose complex functions into sub-modules is that of modularity. A decomposition is considered to attain high modularity when each individual module is internally coherent, and all the modules are loosely coupled with one another. One of our research focuses is to develop methodologies to formalize this notion of modularity quantitatively, and to use it to evaluate a given decomposition, thus enable us to develop systematic techniques for obtaining an optimal functional decomposition of the information management functions. A particularly promising approach that we are actively investigating is the Systematic Design Methodology (SDM) (Huff and Madnick, 1978). The following briefly describes this approach.

The SDM approach to system design centers on the problem of identifying a system's modules, or "sub-problems", their functions, and their interconnections. Using the SDM approach, we begin with a set of functional requirement statements for the INFOPLEX information management functions. Each pair of requirements is examined in turn, and a decision as to whether a significant degree of interdependence between the two requirements exists is made. Then the resulting information is represented as a non-directed graph structure: nodes are requirement statements, links are assessed interdependencies. The graph is then partitioned with the objective of locating a good

decomposition. An index of partition goodness is employed, which incorporates measures of subgraph "strength" and "coupling." The actual goodness index is taken as the algebraic difference between the strengths of all the subgraphs, and the inter-subgraph couplings. That is, M=S-C, where S is the sum of the strength measures of all subgraphs, and C is the sum of all the inter-subgraph couplings.

Once an agreeable partition is determined, the resulting sets of requirements are interpreted as "design sub-problems." From these design sub-problems a functional hierarchy of the INFOPLEX can then be systematically derived. This procedure is illustrated in Figure 15. For details of this approach, refer to (Huff and Madnick, 1978).

3.2 Physical Decomposition

As we have discussed in the previous section, the information management functions of INFOPLEX are implemented as a functional hierarchy, using microprocessors. This systematic and modular approach entails highly parallel operations and highly reliable information management modules. To provide a high performance, highly reliable, and large capacity storage system, INFOPLEX makes use of an automatically managed memory hierarchy (referred to as the

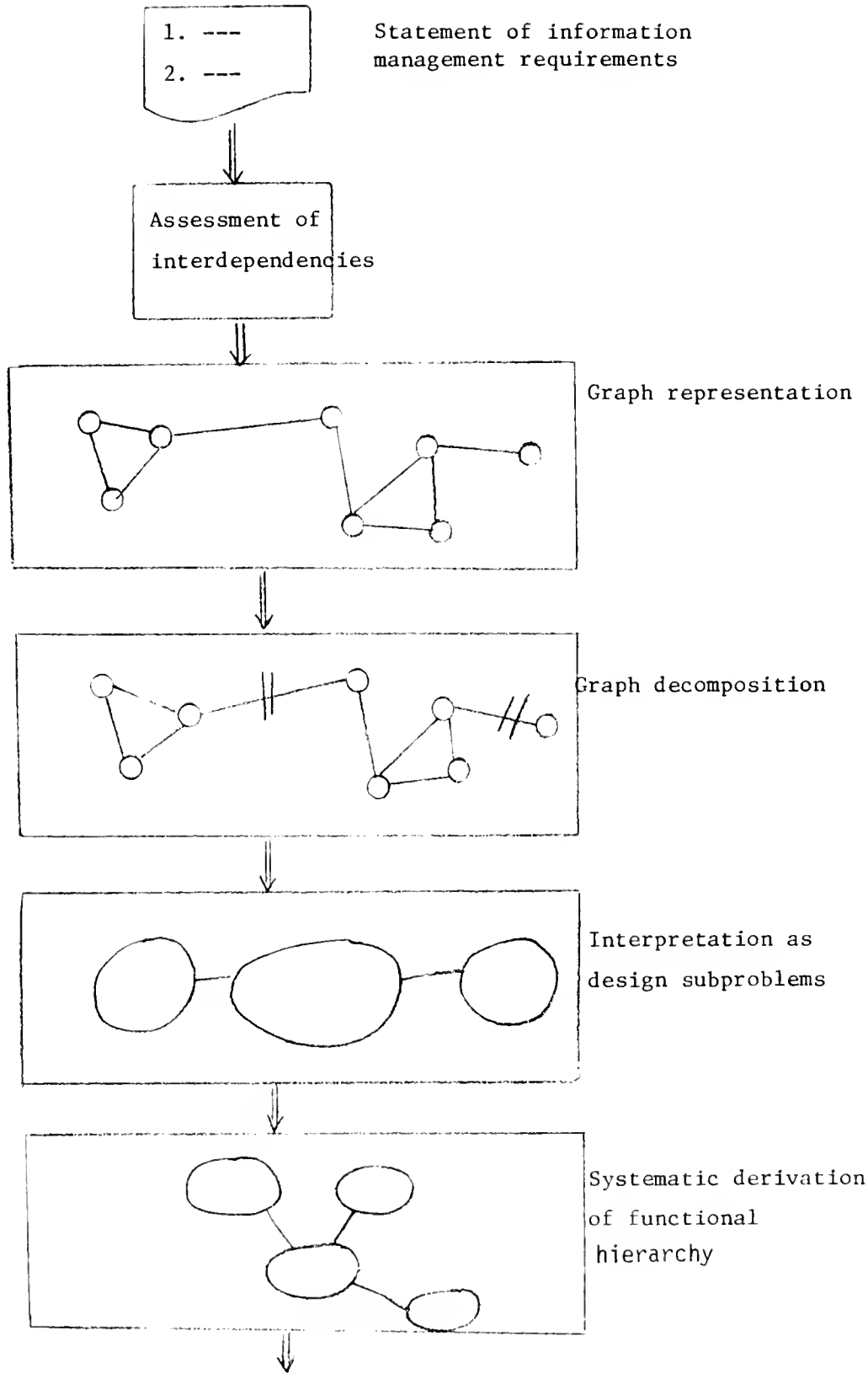

Figure 15. Use of SDM for INFOPLEX functional decomposition

INFOPLEX physical decomposition). In this section, the rationale for and an example of an automatic memory hierarchy are discussed. Then the INFOPLEX approach to realize such a memory hierarchy is also discussed.

3.2.1 Rationale for a memory hierarchy

The technologies that lend themselves to low cost-per-byte storage devices (and, thereby, economical large capacity storage) result in relatively slow access times. If it was possible to produce ultra-fast limitless-capacity storage devices for miniscule cost, there would be little need for a physical decomposition of the storage. Lacking such a wondrous device, the requirements of high performance at low cost are best satisfied by a mixture of technologies combining expensive high-performance devices with inexpensive lower-performance devices.

There are many ways that such an ensemble of storage devices may be structured, but in our research (Madnick, 1973; Madnick, 1975a) we have found the technique of hierarchical physical decomposition to be very effective. Using this technique, the ensemble of heterogeneous storage devices is organized as a hierarchy. Information is moved between storage levels automatically depending upon actual or anticipated usage such that the information most likely to be referenced in the future is kept at the highest (most

easily accessed) levels.

The effectiveness of a memory hierarchy depends heavily on the phenomonon known as locality of reference (Denning, 1970; Madnick, 1975a). A memory hierarchy makes use of this property of information reference pattern so that the information that is used frequently would be accessible through the higher levels of the hierarchy, giving the memory hierarchy an expected access time close to that of the access time of the faster memories. This approach has been used in contemporary computer systems in cache memory systems (Conti, 1969), in virtual memory demand paging systems (Bensoussan et al., 1969; Chu and Opderbeck, 1974; Greenberg and Webber, 1975; Hatfield, 1972; Mattson et al., 1970; Meade, 1970; Ohnigian, 1975), and in mass storage systems (Considine and Weis, 1969; Johnson, 1975).

Several measures of database locality and experimentations with these measures are reported in a recent study (McCabe, 1978). The observations from these experiments are encouraging. In particular they indicate that there is considerable locality of database reference. As can be inferred from Figure 16, there are a large number of instances where a record that has just been referenced is referenced again very soon. For example, point X in Figure 16 represents the 175 instances where the same record is referenced again after only 5 references were made to other

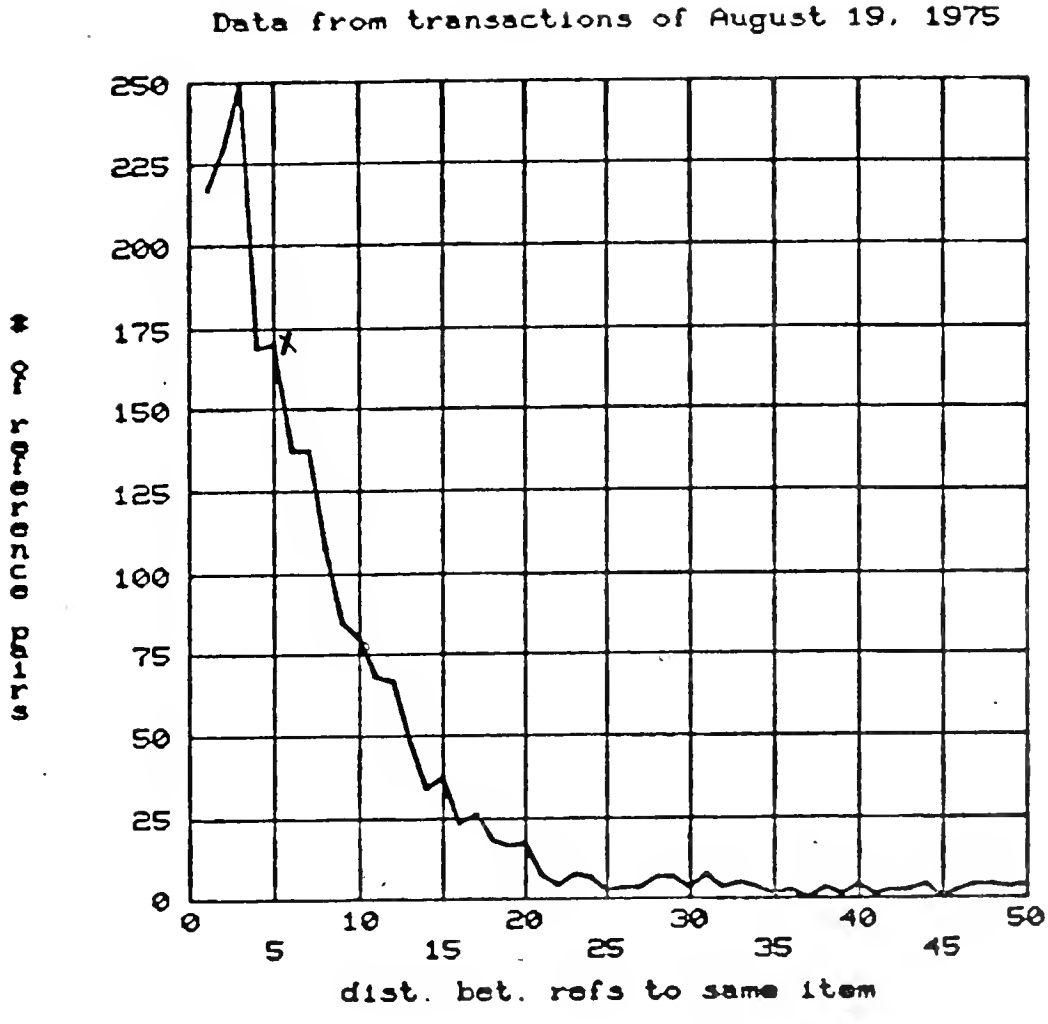

Figure 16. Inter-reference distance in a database system

records.

McCabe's study gives further support for the use of a memory hierarchy as an effective medium for storing very large databases, as is required in INFOPLEX.

3.2.2 Example of a physical decomposition

We now discuss an example of a memory hierarchy, its general structure, types of storage devices that it may employ, and some strategies for automatic information movement in the hierarchy.

3.2.2.1 General structure

To the user (i.e. the lowest level of the functional hierarchy) of the memory hierarchy, the memory appears as a very large linear virtual address space with a small access time. The fact that the memory is actually a hierarchy or that a certain block of information can be obtained from a certain level is hidden from the memory user. Figure 17 illustrates the general structure of a memory hierarchy consisting of six levels of storage devices. Some of the devices that can be used in these levels are discussed in the next subsection.

The lowest level always contains all the information of

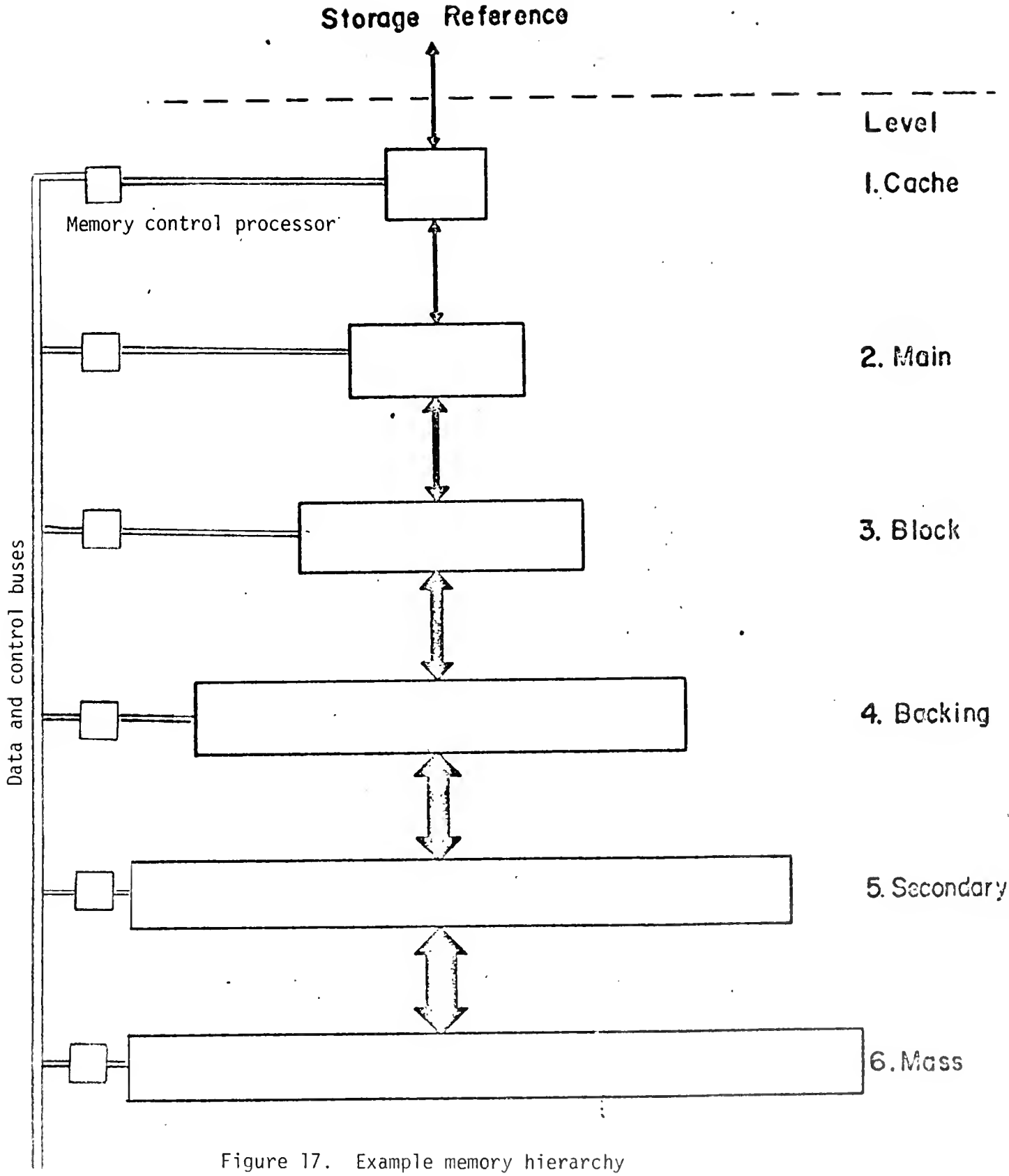

Figure 17. Example memory hierarchy

the system. A high level always contains a subset of the information in the next lower level. To satisfy a request, the information in the highest (most easily accessed) level is used.

Storage reference is accomplished by supplying the memory hierarchy with a virtual address (say a 64-bit address), the memory hierarchy will determine where the addressed information is physically located. The addressed information will be moved up the memory hierarchy if it is found in other than the highest level of the hierarchy. This implies that there is a high variance in the access time of the memory system. This situation is alleviated by providing multiple ports to the memory system so that a pipeline of requests can be processed. Furthermore, the inherent parallelism within each memory level and among different memory levels provides high throughput for the memory system as a whole. Since the functional levels are designed with high parallelism of operation as one of its major objectives, the processor making the request can take advantage of the high memory access time variance. For example, by making use of the expected time of response (ETR) for a given memory request, generated by the memory system in response to a memory request, the processor making the request can schedule its activities accordingly. Various schemes are used to make the automatic management of the memory hierarchy efficient. Some of these strategies

are discussed in a latter section.

3.2.2.2 Storage devices

Traditionally, computer direct access storage has been dominated by two fairly distinct technologies: (1) ferrite core and, later, metal oxide semiconductor (MOS) LSI memories with microsecond access times and relatively high costs, and (2) rotating magnetic media (magnetic drums and disks) with access time in the range of 10 to 100 milliseconds and relatively low costs. This has led to the separation between main storage and secondary storage depicted in Figure 18.

Recently several new memory technologies, most notably magnetic bubbles, electron beam addressed memories (EBAM), and charge coupled devices (CCD), have emerged to fill the "gap" between the two traditional memory technologies. The characteristics of these three particular technologies are summarized in Table 1.

The evolution and increasing deployment of the above and many other memory technologies have produced a more continuous cost-performance range of storage devices, as depicted in Table 2 (Madnick 1975a; Martin and Frankel 1975; Jyers 1976; Wendley 1975). Note that these technologies, which are arbitrarily grouped into six categories, result in

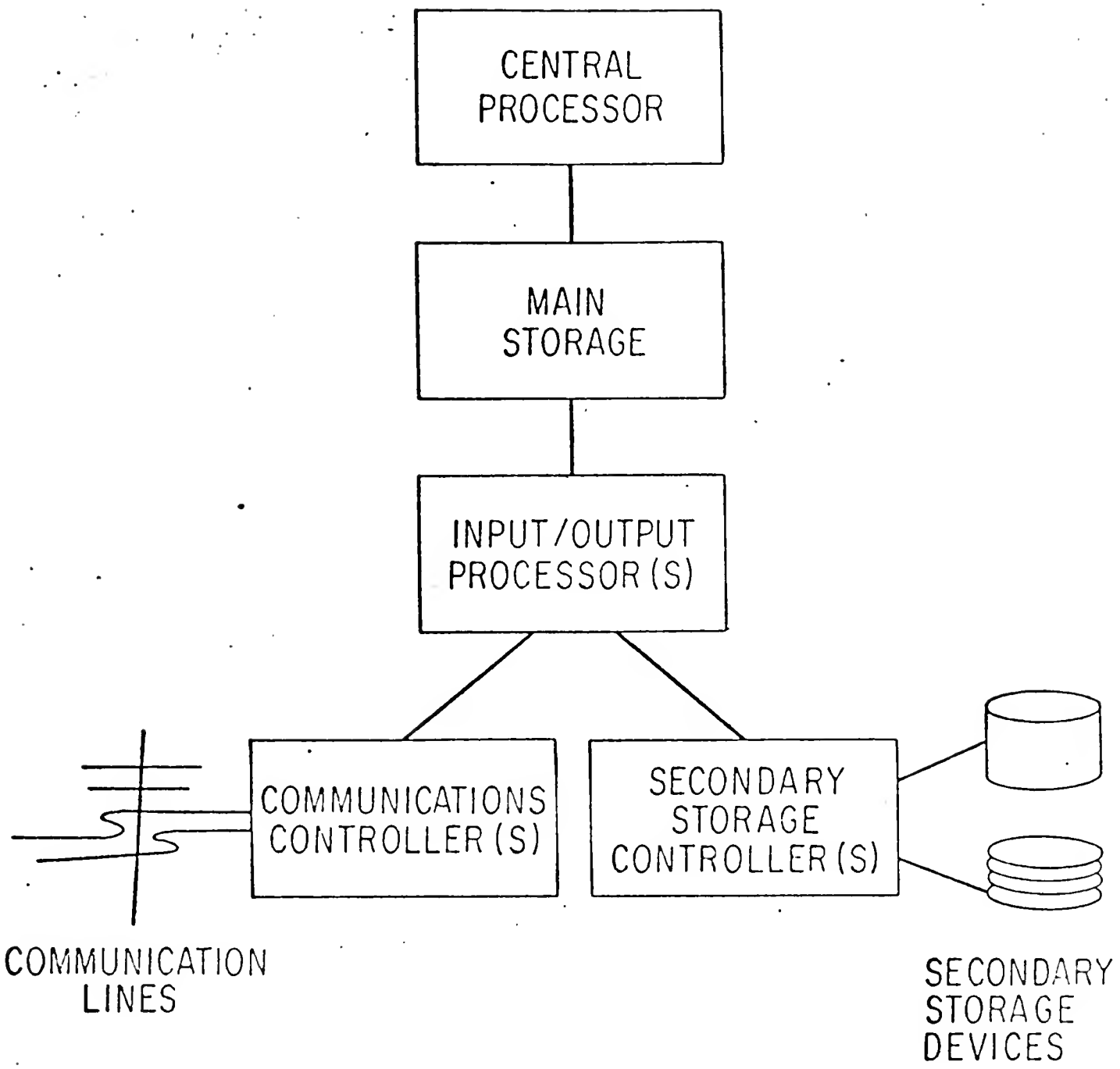

Figure 18. Traditional computer memory organization

Memory Characteristics	Magnetic Bubble	EBAM	CCD
1. Storage Representation	Magnetic Domain	Electrical Charge	Electrical Charge
2. Bit density	10^6 bits per sq. in.	10^7 bits per sq. in.	10^6 bits per sq. in.
3. Access Mode	Sequential	Quasi-Random	Sequential or Block Addressable
4. Access Time	.5-10 msec	3-30 μsec	Sequential: 5-50 μsec Block Access: 5 μsec
5. Shift Rate	100-500Kbs	N/A	1-10 Mbits per sec
6. Transfer Rate	100-500Kbs	1-10 Mbits per sec	1-10 Mbits per sec
7. Power per bit (during memory operation)	2μw	10μw	50μw
8. Cost per bit	.02¢	.005¢	.05¢
9. Largest chip size fabricated	100 Kbit (Rockwell)	32 Mbit (Tube) (GE)	64 Kbit (Mnemonics)
10. Organization	Major-minor loop	Stand-alone or shared electronics mode	Serpentine Serial-Parallel-Serial, Block oriented

TABLE 1 Characteristics of Magnetic Bubble Memories, EBAMs and CCDs

	Storage Level	Random Access Time	Sequential Transfer Rate (bytes/sec)	Unit Capacity (bytes)	System Price (per byte)
1.	Cache	100 ns	100M	32K	50¢
2.	Main	1 us	16M	512K	10¢
3.	Block	100 us	8M	2M	2¢
4.	Backing	2 ms	2M	10M	0.5¢
5.	Secondary	25 ms	1M	100M	0.02¢
6.	Mass	1 sec.	1M	100B	0.0005¢

TABLE 2

Range of Storage Devices

storage devices that span more than six orders of magnitude in both random access time (from less than 100 nanoseconds to more than 1 second) and system price per byte (from more than 50 cents per byte to less than 0.0005 cent).

This evolution has facilitated the choice of appropriate cost-effective storage devices for the memory hierarchy (Boyd 1978; Kluge, 1978). For example, for the memory hierarchy discussed in the previous section, we might use a device like the IBM 3850 Mass Storage as the mass storage, traditional moving head disks as secondary storage, magnetic drums as backing store, CCD or magnetic bubble as block store, core or semiconductor RAM as main storage, and high performance semiconductor RAM as cache.

3.2.2.3 Strategies for information movement

Various physical storage management and movement techniques, such as page splitting, read through, and store behind, can be distributed within the hierarchy of levels. This facilitates parallel and asynchronous operation in the hierarchy. Furthermore, these approaches can lead to greatly increased reliability of operation. For example, under the read through strategy (Figure 19), when data currently stored at level i (and all lower performance levels) is referenced, it is automatically and simultaneously copied and stored into all higher performance

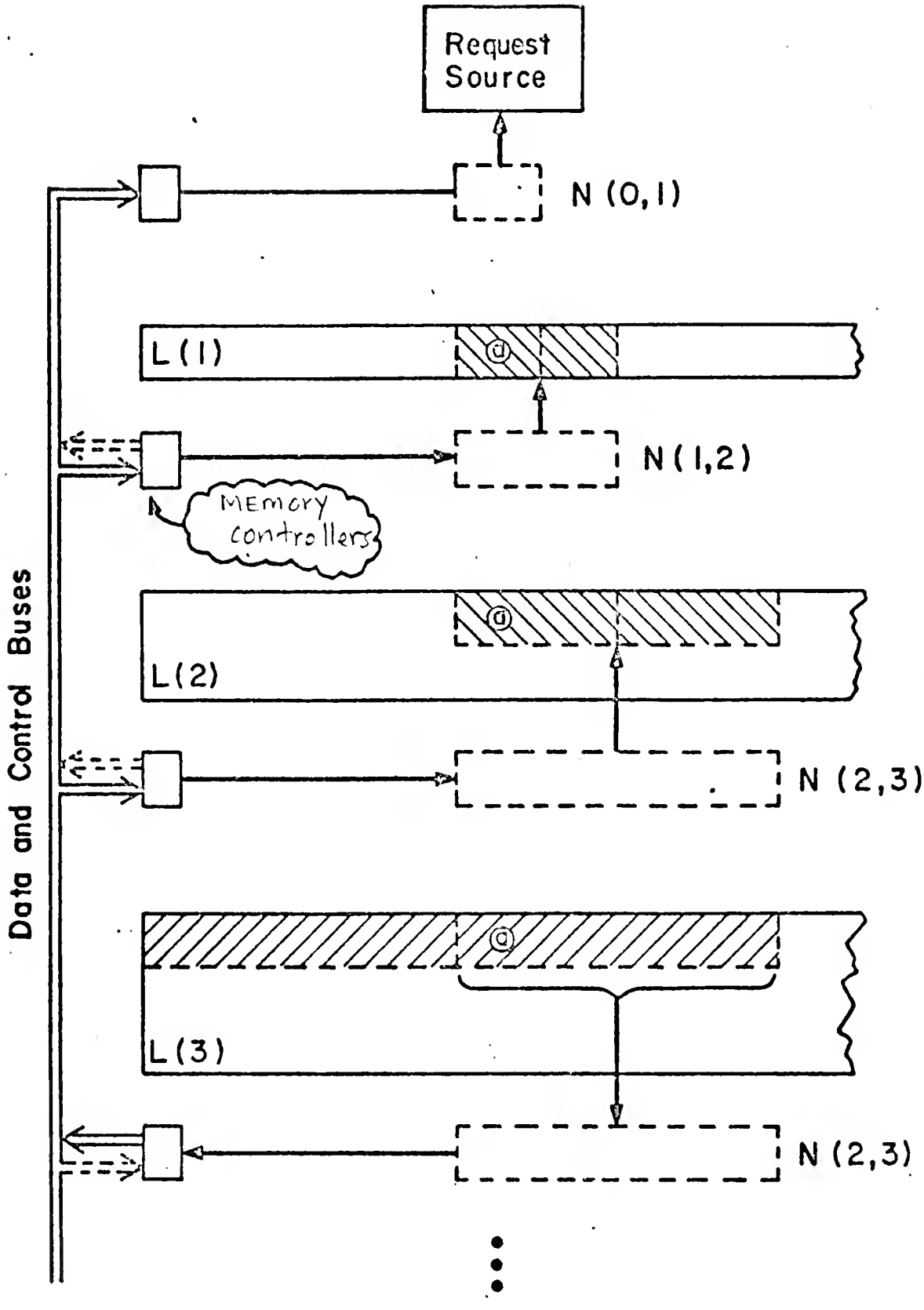

Figure 19. Read-through data movement strategy

levels. The data itself is moved between levels in standard transfer units, also called pages, whose size N (i-1, i) depends upon the storage level from which it is being moved.

For example, suppose that the datum "a", at level 3, is referenced (see Figure 19). The block of size N(2,3) containing "a" is extracted and moved up the data bus. Level 2 extracts this block of data and stores it in its memory modules. At the same time, level 1 extracts a sub-block of size N(1,2) containing "a" and level Ø extracts the sub-block of size N(Ø,1) containing "a" from the data bus.

Hence, under the read through strategy, all upper storage levels receive this information simultaneously. If a storage level must be removed from the system, there are no changes needed. In this case, the information is "read through" the level as if it didn't exist. Since all data available at level i is also available at level i + 1 (and all other lower performance levels), there is no information lost. Thus, no changes are needed to any of the other storage levels or the storage management algorithms although we would expect the performance to decrease as a result of the missing storage level. A limited form of this reliability strategy is employed in most current-day cache memory systems (Conti, 1969).

In a store behind strategy all information to be changed is first stored in L(1), the highest performance storage level. This information is marked "changed" and is copied into L(2) as soon as possible, usually during a time when there is little or no activity between L(1) and L(2). At a later time, the information is copied from L(2) to L(3), etc. A variation on this strategy is used in the MULTICS Multilevel Paging Hierarchy (Greenberg and Webber, 1975). This strategy facilitates more even usage of the bus between levels by only scheduling data transfers between levels during idle bus cycles. Furthermore, the time required for a write is only limited by the speed of the highest level memory.

The store behind strategy can be used to provide high reliability in the storage system. Ordinarily, a changed page is not allowed to be purged from a storage level until the next lower level has been updated. This can be extended to require two levels of acknowledgment. Under such a strategy, a changed page cannot be removed from L(1) until the corresponding pages in both L(2) and L(3) have been updated. In this way, there will be at least two copies of each changed piece of information at levels L(i) and L(i+1) in the hierarchy. Other than slightly delaying the time at which a page may be purged from a level, this approach does not significantly affect system performance. As a result of this technique, if any level malfunctions, it can be removed

from the hierarchy without causing any information to be lost. There are two exceptions to this process, L(1) and L(n). To completely safeguard the reliability of the system, it may be necessary to store duplicate copies of information at these levels only.

Figure 20 illustrates this process. In Figure 20(a), a processor stores into L(1), the corresponding page is marked "changed" and "no lower level copy exists". Figure 20(b) shows in a latter time, the corresponding page in L(2) is updated and marked "changed" and "no lower level copy exists". An acknowledgment is sent to (L1) so that the corresponding page is marked "one lower level copy exists". At a latter time (Figure 20(c)), the corresponding page in L(3) is updated and marked "changed" and "no lower level copy exists". An acknowledgment is sent to L(2) so that the corresponding page is marked "one lower level copy exists". An acknowledgment is sent to L(1) so that the corresponding page is marked "two lower level copy exists". At this time, the page in L(1) may be replaced if necessary, since then there will be at least two copies of the updated information in the lower memory levels.

3.2.3 INFOPLEX's approach to physical decomposition

In the previous section, we have illustrated an example of a memory hierarchy that makes use of an ensemble of

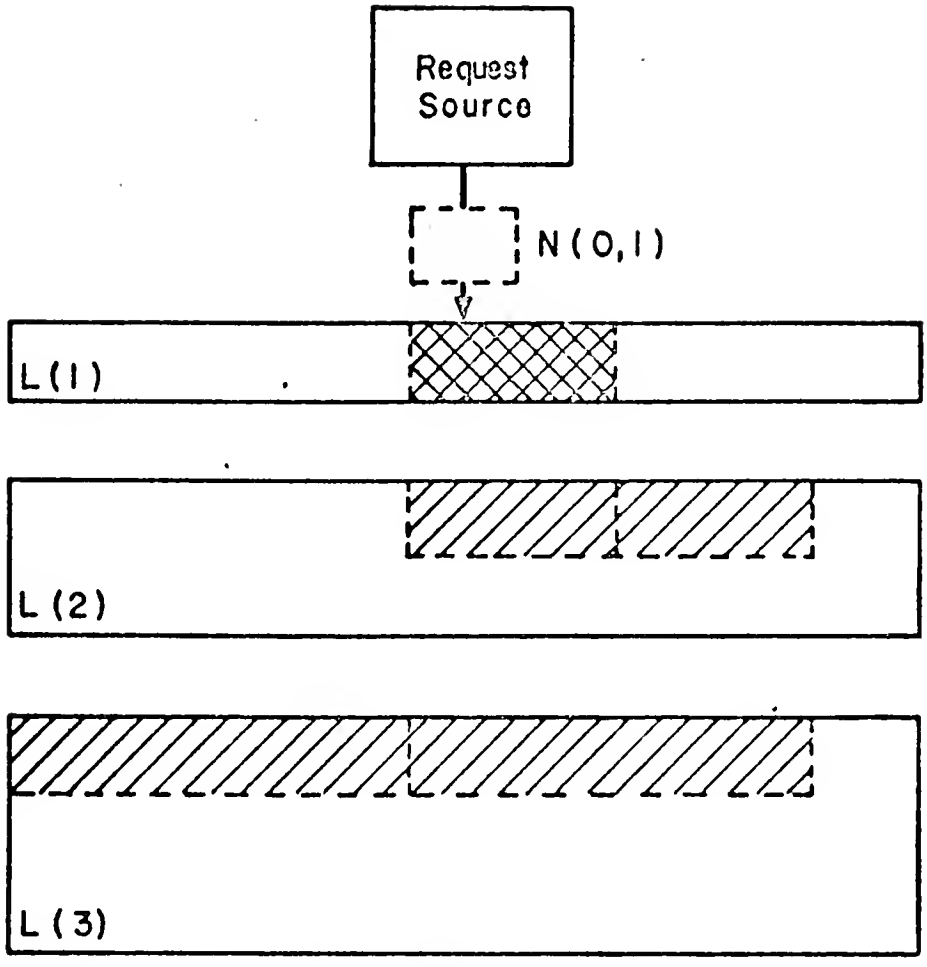

Figure 20(a) Store-behind (a)

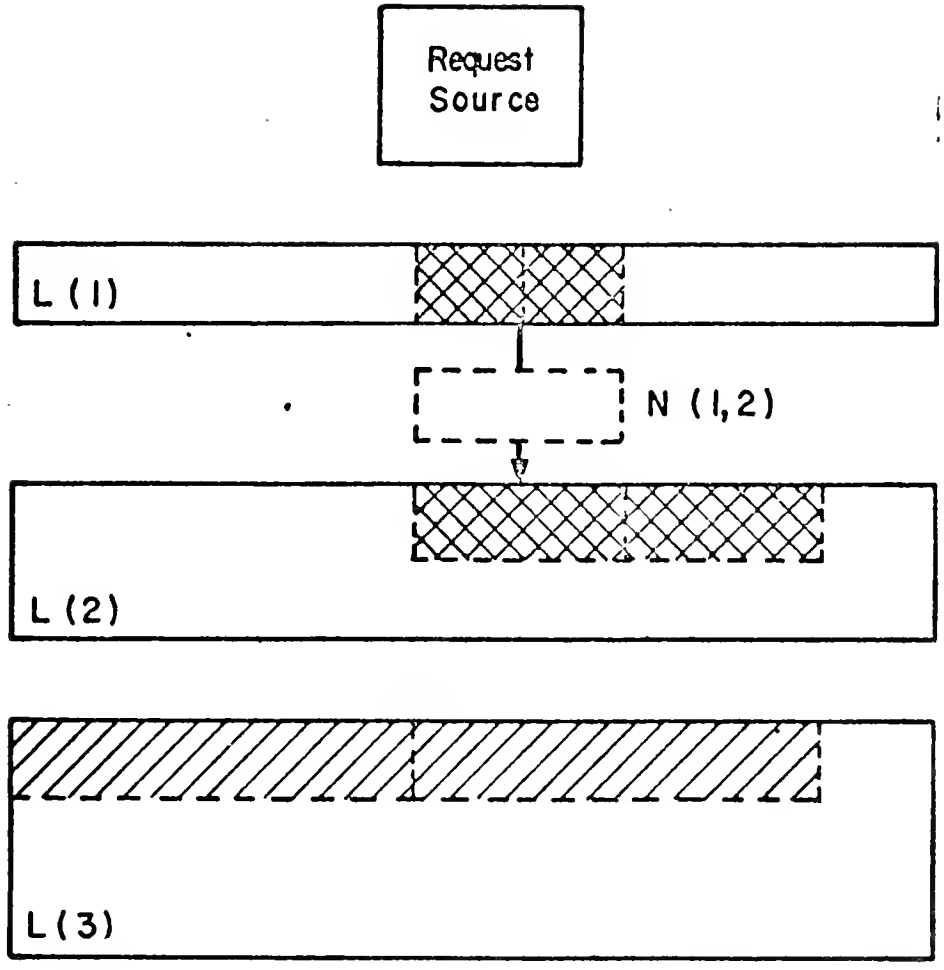

Figure 20(b) Store-behind (b)

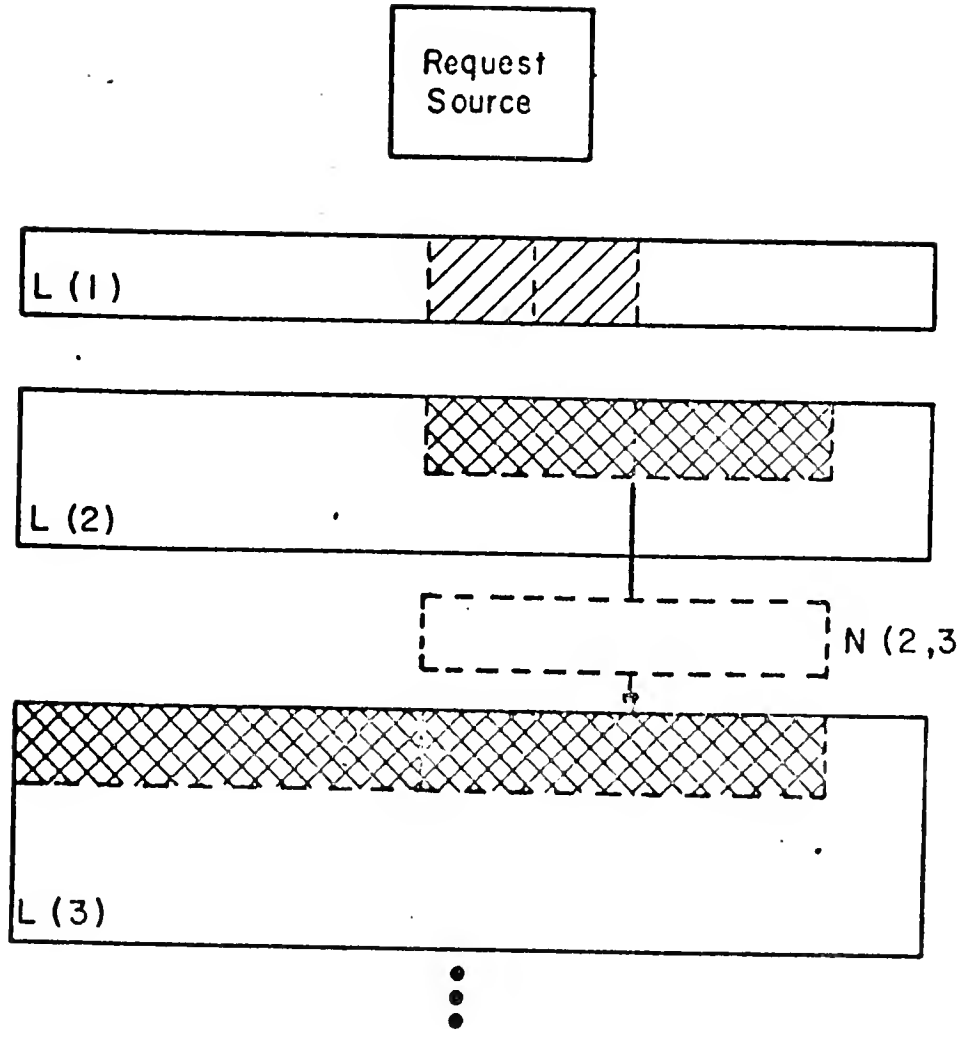

Figure 20(c) Store-behind (c)

heterogenous storage devices. Although memory hierarchies using two or three levels of storage devices have been implemented, no known generalized automatic memory hierarchy has been developed.

The optimality of a memory hierarchy depends on the complex interactions among the memory reference pattern, the device characteristics, and the information movement strategies. Our approach to this complex problem is to empirically study and characterize data reference patterns at several levels (e.g. transaction level, logical data level, and physical data level), to develop various information movement strategies, and to design a prototype memory hierarchy. The interactions among these components are then to be systematically investigated by means of analytic models and simulation models.

3.3 Distributed control and multiple-microprocessor implementations

Both the INFOPLEX functional and physical decomposition make use of distributed control mechanisms and multiple microprocessors in their implementations. In the functional decomposition, multiple microprocessors are used to implement the information management functions as well as the distributed control mechanisms. In the physical decomposition, multiple microprocessors are used mainly to

implement the distributed control mechanisms. The following briefly illustrate these ideas.

3.3.1 Functional hierarchy implementation

As noted earlier, each level of functional decomposition is implemented in terms of the primitives of the next lower level (refer to Figures 8 and 9). A separate set of microprocessors is used to implement each functional level. Thus, multiple requests can be at different stages of processing at different levels simultaneously. A view of the multiple processor implementation of the hierarchical functional decomposition is depicted in Figure 21. Each processor level implements a particular information management function. For example, while one request is being checked for access rights by the Data Verification and Access Control Module (Level 6), an earlier request may be undergoing inter-relation navigation in the Relation Linkage Processor Module (Level 4).

Furthermore, multiple identical modules at each level can be used so as to enhance parallelism of operation and high reliability at each level. For example, Level 6 in Figure 21, the Data Verification and Access Control Level, may be implemented using three identical processors each capable of performing the same security checking functions (See Figure 22). All the processors can operate in parallel. Thus,

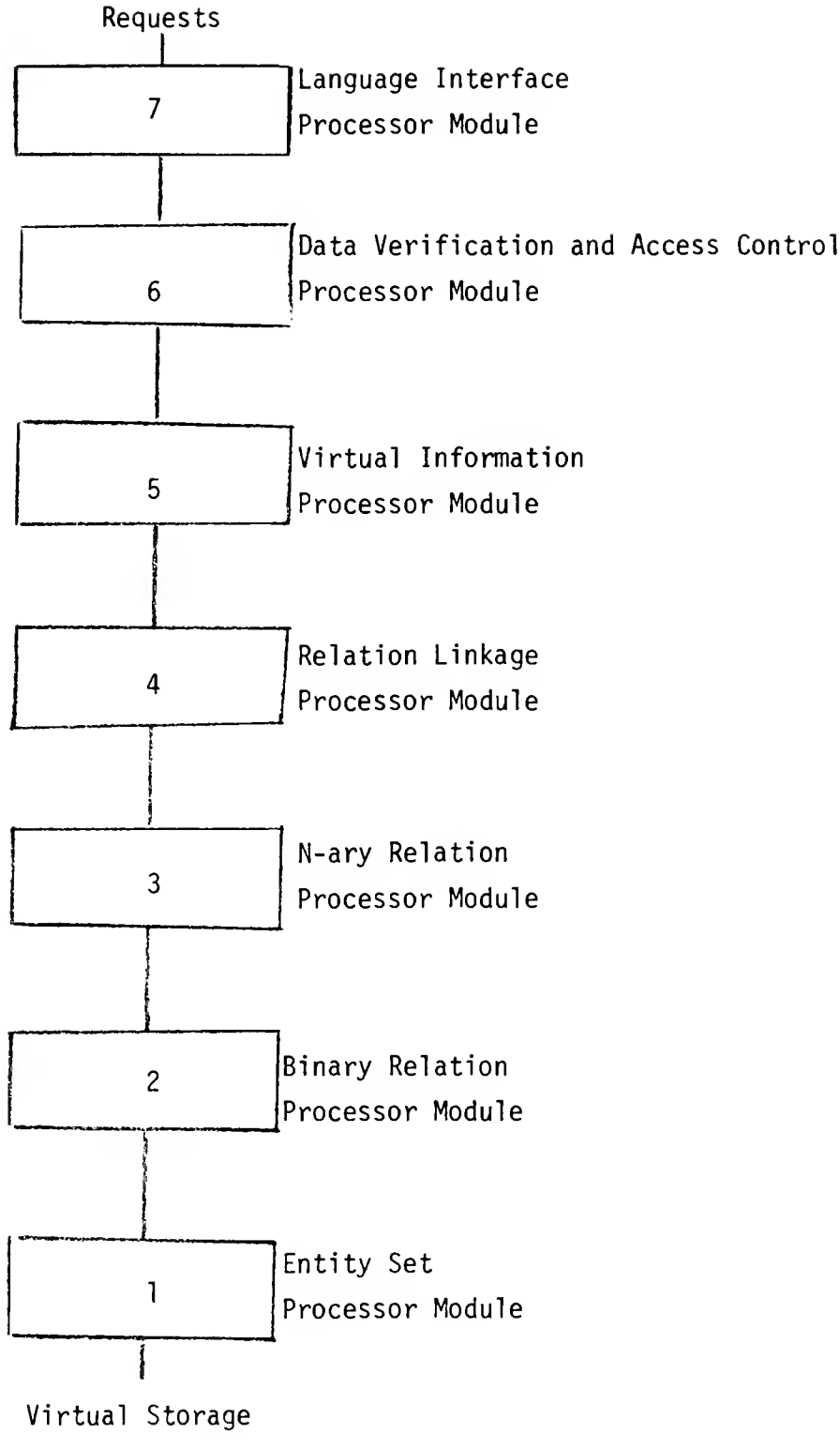

Figure 21. Multiple Microprocessor Implementation of Example Functional Hierarchy

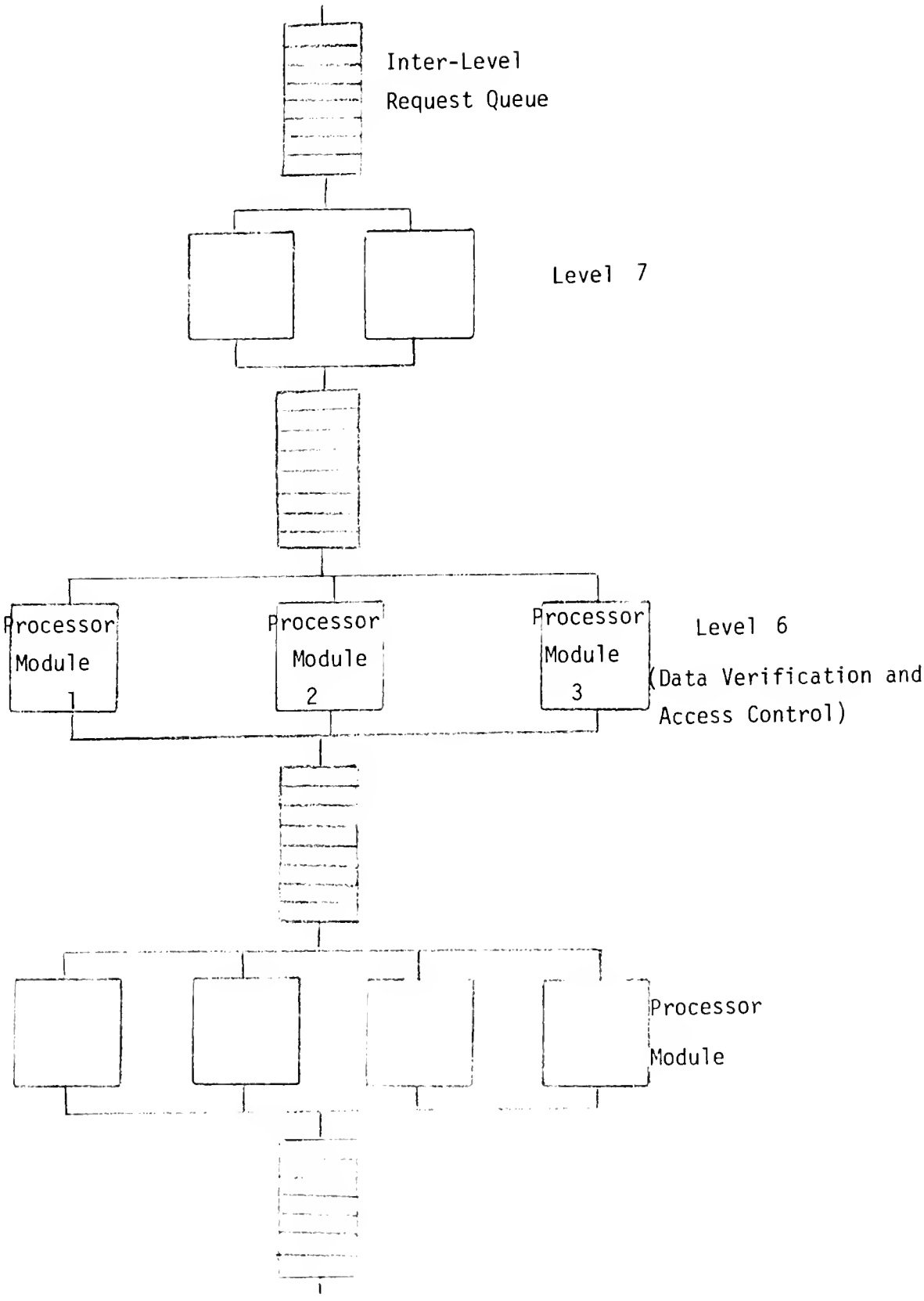

Figure 22. Multiple Identical Processor Module at Each Functional Level

three different requests can be checked for access rights at the same time, each using a different Data Verification and Access Control Level processor. Using such multiple interchangable processors at each level enhances reliability since a malfunctioning processor can be removed from service, and the system can still operate uninterrupted, in a somewhat degraded mode, using the remaining processors that perform the same function.

The inter-level queues maintain a pipeline of requests to each functional level. All levels can be simultaneously processing requests. This parallelism of operation among different functional levels together with the parallelism of operation within each functional level can provide very high throughput.

Although such extensive use of processors has been quite expensive in the past, the advent of low-cost microprocessors makes such a system economically feasible (Weissberger, 1977). Furthermore, since each level implements only a limited amount of the total system's functionality, very simple processors can be used.

3.3.2 Physical hierarchy implementation

As illustrated in the sample memory hierarchy (Refer to Figure 17), a possible architecture of the memory hierarchy

is to have data and control buses connecting each memory level. The same philosophy as in the functional hierarchy is employed to enhance system throughput in the memory hiearchy. Simultaneous operation of all the memory levels is possible via pipelining of memory requests at each memory level. Parallel operation at each memory level is possible using multiple memory modules (using some form of interleaving).

The memory controllers at each memory level realize the various distributed control functions of the memory hierarchy. For example, a memory controller may consist of a microprocessor complex that implements the various data transfer algorithms between adjacent memory levels. A memory controller may also incorporate an associative memory for mapping of virtual addresses into their real counterparts. Using such an associative memory, a memory controller can quickly determine if the addressed information is to be found in the particular memory module.

3.3.3 Distributed Control

The INFOPLEX functional hierarchy and physical hierarchy, as described above, are based upon distributed control algorithms. That is, each processor module only interacts with neighboring processor modules (i.e., the next higher and the lower level of the hierarchy). There is no "central

control" processor that must control all other processors and the data flow. This approach eliminates potential bottlenecks and the existence of critical reliability elements in the system. These and other major advantages of the INFOPLEX architecture are summarized in the next section.

3.4 Advantages of the INFOPLEX Architecture

Compared with conventional computer architectures, the INFOPLEX architecture has the following advantages:

(1) It is able to handle extremely high request rates since it takes advantage of pipelining and parallel processing in its architecture.

(2) Its hierachical decomposition has been explicitly designed for information management. The hierachical decomposition theory has been successfully used by ourselves and others in the design of operating systems, file systems, and database systems. Thus hierarchical decomposition is not only theoretically sound but also empirically tested.

(3) It utilizes a highly decentralized control mechanism to eliminate any potential bottlenecks that may

occur due to centralized control.

(4) It provides both extremely high reliability. In particular,

a. the reliability is attained by exploiting the particular characteristics of the memory rather than brute force redundancy;

b. the system can operate with any single failure of any processor or memory device, and no information is lost or misused; it can also operate in spite of most double or triple failures; and

c. as a result of the above, no manual intervention is required to cope with such failures.

(5) It can integrate new memory technologies easily by adding or replacing a memory level in the memory hierarchy.

(6) It explicitly provides multiple interfaces to different types of users, such as end users, system analysts, system programmers and hardware maintainers.

4. RESEARCH DIRECTIONS

We plan to investigate the following major research topics.

4.1 Optimal Hierarchical Functional Decomposition

Hierarchical decomposition has been used successfully in designing file systems (Madnick and Alsop, 1969), operating systems (Madnick and Donovan, 1974; Madnick, 1976), and database systems (Donovan, 1975). Similar techniques have also been used by other researchers in systems design (Dijkstra, 1968; Parnas, 1974). However, the decomposition of functionalities into different modules, whether they are hardware or software, is usually based on the designer's experience and intuition. The resulting modules may not be optimal. Therefore, we plan to develop analytical tools to be used in the INFOPLEX architectural design process.

One of the techniques we are investigating is the requirement decomposition method (Alexander, 1964; Andreu and Madnick, 1977), and the Systematic Design Methodology (SDM) (Huff and Madnick, 1978). Using the SDM approach (refer to figure 13.1 in a previous section), the functional requirements of the INFOPLEX database computer are modeled as nodes in a graph, and a link between two nodes indicates that the corresponding functionalities are related to each

other. The problem is to separate the graph into subgraphs such that each subgraph is loosely coupled to other subgraphs and strongly coherent. Each of the resultant subgraphs corresponds to a module of the system. Many graph partition algorithms exist (Hartigan, 1975), including an algorithm we developed for putting highly correlated program segments into pages (Chen and Gallo, 1974). We shall study ways in which to apply these algorithms to our problem of hierarchical functional decomposition. Specific issues to be addressed include: (1) What are the objective function and constraints of the graph partition problem in this context? (2) What is a precise definition of the "link" between two nodes? (3) How to systematically obtain a functional hierarchy from the set of subgraphs?

The ultimate goal of this phase of the research is to develop a tool that can help define the optimal INFOPLEX hierarchical functional decomposition.

4.2 Optimal Hierarchical Physical Decomposition

The memory hierarchy in INFOPLEX is flexible in incorporating new technologies and dropping old ones. A key design problem in a memory hierarchy is to decide how many memory levels (i.e., what types of memory technologies) are needed and what the size of each level should be. Some researchers (Arora and Gallo, 1971; Ramamoothy and Chandy,

1970) have studied the criteria for admission of a new memory level (with given cost per unit storage and access time) to the memory hierarchy. However, they did not consider queueing delays, the read-through, and the write-behind operation.

As the INFOPLEX memory hierarchy is primarily used for storage of very large databases, the nature of database references has important implications on the optimality of the physical decomposition. For example, we have observed that database references exhibit localities of different durations and periodicities (e.g., certain portions of a payroll database may be heavily accessed, but this access pattern may only occur weekly). An optimal physical decomposition will have to provide cost effective memory levels to store information that is referenced with different frequencies and locality patterns. We plan to perform experimental and analytical studies of database reference characteristics and to derive a scheme to optimize the hierarchical physical decomposition of the memory hierarchy that takes account of all the above additional considerations as well as the special characteristics of INFOPLEX.

The memory control unit at each level of the memory hierarchy will be studied in detail including the various algorithms for information transfer among memory levels,

mechanisms for virtual address mapping using associative memories and the communications protocols among these distributed controls. An important research issue will be the reliability implications and the degree of parallelism realizable using these mechanisms. A preliminary design of the INFOPLEX memory hierarchy is currently being used to explore these issues.

4.3 Multiple-Microprocessor Architecture and Protocols

The modules of the hierarchical functional decomposition and hierarchical physical decomposition must be realized using sets of microprocessors. It is necessary to identify those characteristics that would be best suited to implementing these modules in order to select the appropriate type of microprocessors. The inter-processor protocols must also be defined in more detail, with special concern for high performance and high reliability. A variety of interconnection strategies must be investigated, including: (1) shared buses, (2) segmented buses, (3) inter-bus queues, and (4) "pended" buses.

4.4 Performance Evaluation and Optimization

It is desirable to know how much can be gained in performance by use of the INFOPLEX architecture in comparison with conventional computer architectures. In

other words, we wish to know what the expected performance of the INFOPLEX architecture would be. In addition, we need to formulate a set of decision rules that can be used by the INFOPLEX modules to automatically select the "best" algorithm, from among a set of available algorithms, to accomplish a function under differing circumstances. Therefore, a quantitative model of INFOPLEX is needed to predict the performance and evaluate different design and algorithm alternatives.

We plan to model INFOPLEX as a hierarchy of performance submodels, each level in the hierarchical decomposition having a performance submodel. Many of these submodels have already been developed by researchers in the field (Severance et al., 1976; Cardenas, 1975; Rothnie, 1975; Chen, 1975; Chen, 1976; Sarmento and Chen, 1977). What needs to be done is to select appropriate models for each level and organize them in a uniform manner.

4.5 Reliability

There are two major types of reliability concerns in the INFOPLEX design. One is related to the reliability of the functional decomposition, and the other to the reliability of the physical decomposition.

We have described how at each hierarchical functional

level multiple microprocessors will be used in parallel to process data requests. As a result, the system is composed of a series of parallel microprocessors. It is desirable to develop a reliability model that will allow us to estimate the effective reliability of this microprocessor complex under various failure assumptions.

In the physical decomposition's memory hierarchy, the failure of a single memory level will not disrupt the operations of the memory hierarchy. However, we would like to be able to model the reliability of the system to answer questions, such as what the probability is of information loss due to a simultaneous failure in two portions of the memory hierarchy.

In addition to modeling the reliability of the system, we shall also investigate ways to improve the reliability of the system.

4.6 Interlocks

Various interlock mechanisms must be used in an information system to coordinate update operations. In designing INFOPLEX, it is necessary to develop interlock techniques that lend themselves to a highly decentralized implementation without adversely affecting performance or reliability. A variety of interlock strategies have been

proposed in the literature and used in current-day database systems. We plan to investigate these strategies in light of the special characteristics of the INFOPLEX architecture to determine the most appropriate approaches for such a highly parallel and decentralized system.

5. CONCLUSIONS

The need for highly reliable information management systems that can support transaction volume and storage capacity several orders of magnitude higher than current information systems calls for new architectures, both in information management functions and the hardware that supports these functions.

The INFOPLEX Data Base Computer architecture is a major step in this direction. INFOPLEX makes use of the theory of hierarchical decomposition to realize a highly structured, modular architecture. The information management functions are structured as a hierarchy and a highly modular implementation of the functional hierarchy using multiple microprocessors entails high degrees of parallelism and reliability. An automatic memory hierarchy is used to support the high performance, high capacity memory requirements of INFOPLEX.

This paper discusses concepts of INFOPLEX. The main objectives of INFOPLEX research are to develop an optimum functional hierarchy and an optimum physical (memory) hierarchy. We have discussed our research directions towards these goals.

6. BIBLIOGRAPHY AND REFERENCES

Key to abbreviations used:

ACM : Association for Computing Machinery

AFIPS : American Federation of Information Processing Societies

CACM : Communications of the ACM

FJCC : Fall Joint Computer Conference

IBMSJ : IBM Systems Journal

IBMJRD : IBM Journal of Research and Development

IEEE : Institute of Electrical and Electronic Engineers

IFIP : International Federation of Information Processing

NCC : National Computer Conference

SJCC : Spring Joint Computer Conference

TODS : ACM Transctions on Database Systems

VLDB : Proceedings International Conference on Very Large Data Bases

(Ahearn et. al., 1972)

Ahearn, G.P., Dishon, Y., and Snively, R.N. 'Design Innovations of the IBM 3830 and 2835 Storage Control Units'. IBMJRD, 16, 1 (January 1972), 11-18.

(Andreu and Madnick, 1977)

Andreu, R. and Madnick, S.E. 'A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation'. CISR-99, Sloan School of Management, MIT, March 1977.

(ANSI, 1975)

ANSI 'Interim Report of ANSI/X3/SPARC group on Database Management Systems' ANSI, February, 1975.

(Armenti et. al., 1970)

Armenti, A., Galley, S., Goldberg, R., Nolan, J., and Scholl, A. 'LISTAR - Lincoln Information Storage and Associatve Retrieval System'. AFIP Conference Proceedings, 36, (1970), 313-322.

(Arora and Gallo, 1971)

Arora, S.R., and Gallo, A. 'Optimal Sizing Loading and Reloading in a Multi-level Memory Hierarchy System'. SJCC, 1971, 337-344.

(Astrahan et. al., 1975)

Astrahan, M.M. and Chamberlin, D.C. 'Implementation of a Structured English Query Language'. CACM, 18, 10 (October 1975)

(Astrahan et. al., 1976)

Astrahan, M.M., et. al. 'System R: A Relational Approach to Data Base Management'. TODS, 1, 3, (September 1976), 1-25.

(Bachman, 1975)

Bachman, C. 'Trends in Database Management - 1975'. AFIP Conference Proceedings, 44, (1975), 569-576.

(Baum, et. al., 1976)

Baum, R.I., Hsiao, D.K., and Kannan, K. 'The Architecture of Database Computer - Part I: Concepts and Capabilities'. OSU-CISRC-TR-76-1, The Ohio State University, September, 1976.

(Baum and Hsiao, 1976)

Baum, R.I., and Hsiao, D.K. 'Database Computers - A Step Towards Data Utilities'. IEEE Transactions on Computers, C-25, 12 (December 1976), 1254-1259.

(Bensoussan et. al., 1969)

Bensoussan, A., Clingen, C.T., and Daley, R.C. 'The MULTICS Virtual Memory'. Second Symposium on Operating Systems Principles, Princeton University, October 1969, 30-42.

(Berra, 1974)

Berra, P.B. 'Some Problems in Associative Processor Applications to Data Base Management'. NCC, 43, (1974), 1-5.

(Berra and Singhania, 1976)

Berra, P.B., and Singhania, A.K. 'A Multiple Associative Memory Organization for Pipelining a Directory to a Very Large Data Base'. Digest of Papers COMPCON 76, 109-112.

(Bobeck et. al., 1975)

Bobeck, A.H., Bonyhard, P.I., and Geusic, J.E. 'Magnetic Bubbles - An Emerging New Memory Technology'. Proceedings IEEE, 63, 8 (August 1975), 1176-1195.

(Boyd, 1978)

Boyd, D.L. 'Implementing Mass Storage Facilities in Operating Systems' Computer, February, 1978, 40-45.

(Bryant, 1966)

Bryan, P. 'Levels of Computer Systems'. CACM, 9, 12 (December 1966), 873-878.

(Buzen and Chen, 1974)

Buzen, J.P., and Chen, P.P. 'Optimal Load Balancing in Memory Hierarchies'. Proceedings IFIP Congress 1974, August 1974, Stockholm, Sweden.

(Canaday et. al., 1974)

Canaday, R.H., Harrison, R.D., Ivie, E.L., Ryder, J.L., and Wehr, L.A. 'A Back-end Computer for Database Management'. CACM, 17,10 (October 1974), 575-584.

(Canning 1974)

Canning, R.G. 'Problem Areas in Data Management'. E.D.P. ANALYZER, 12, 3 (March 1974).

(Canning, 1976)

Canning, R.G. 'Distributed Data Systems'. E.D.P. ANALYZER, 14, 6 (June 1976).

(Cardenas, 1973)

Cardenas, A. 'Evaluation and Selection of File Organization - A Model and a System'. CACM, 16, 9 (September 1973), 540-548.

(Cardenas, 1975)

Cardenas, A. 'Analysis and Performance of Inverted Database Structures'. CACM, 18, 5 (May 1975).

(Carrow, 1972)

Carrow, P. 'Machine-independent Data Management Systems'. Proceedings DoD Joint ADP Conference, III, (October 1972), 13.1-13.7.

(Chang, 1975)

Chang, H. 'Capabilities of the Bubble Technology'. NCC, 44, (1975), 847-855.

(Chamberlin et. el., 1975)

Chamberlin, D., Gray, J.N., and Traiger, I.L. 'Views, Authorization and Locking in a Relational Database System'. NCC, 44, (1975).

(Chen, 1973)

Chen, P.P. 'Optimal File Allocation in Multi-level Storage Systems'. NCC, June 1973.

(Chen and Gallo, 1974)

Chen, P.P., and Gallo, A. 'Optimization of Segment Packing in Virtual Memory'. Computer Architecture and Network, E. Gelenbe and R. Mahl, eds., 1974, North Holland.

(Chen, 1975)

Chen, P.P. 'Queueing Network Model of interactive Computing Systems'. Proceedings IEEE, 63, 6 (June 1975), 954-957.

(Chen, 1976)

Chen, P.P. 'The Entity-relationship Model - Toward a Unified View of Data'. TODS, 1, 1 (March 1976).

(Chu and Opderbeck, 1974)

Chu, W.W., and Operbeck, H. 'Performance of Replacement Algorithms with Different Page Sizes'. Computer, 7, 11 (November 1974), 14-21.

(Codasyl, 1971)

Codasyl. 'Data Base Task Group, April 1971 Report'. ACM, New York, 1971.

(Codd, 1970)

Codd, E.F. 'A Relational Model of Data for Large Shared

Data Banks'. CACM, 13, 6 (June 1970), 377-387.

(Codd, 1974)

Codd, E.F. 'Recent Investigations in Relational Database Systems'. Information Processing 71, North Holland Publishing Company, 1974.

(Cohen and Chang, 1975)

Cohen, M.S. and Chang, H. 'The Frontier of Magnetic Bubble Technology'. Proceedings IEEE, 63, 8 (August 1975), 1196-1206.

(Computer Science Corp., 1974)

Computer Science Corp. 'USN/USMC Future Data Systems Requirements and Their Impact on the AADC'. Report prepared for the US Naval Electronics Systems Command, 1974.

(Considine and Weis, 1969)

Considine, J.P., and Weis, A.H. 'Establishment and Maintenance of a Storage Hierarchy for an On-line Database Under TSS/360'. FJCC, 35 (1969), 433-440.

(Conti, 1969)

Conti, C.J. 'Concepts for Buffer Storage'. IEEE Computer Group News, March 1969, 6-13.

(Conway and Snigier, 1976)

Conway, J., and Snigier, P. 'Second Annual Microcomputer Systems Directory'. EDN, 21, 21 (Novemeber 20, 1976), 95-123.

(Copeland et. al., 1973)

Copeland, G.P., Lipovski, G.J., and Su, S.Y.W. 'The Architecture of CASSM: A Cellular System for Non-numeric Processing'. Proceedings of First Annula Symposium on Computer Architecture, December, 1973, 121-128.

(Coulouris et. al., 1972)

Coulouris, G.F., Evans, J.M., and Mitchell, R.W. 'Towards Content Addressing in Data Bases'. Computer, 15, 2 (February, 1972), 95-98.

(Cox, 1972)

Cox, R.A. 'Intelligent Support Systems'. Proceedings DoD Joint ADP Conference, III, (October, 1972), 15.1-15.10.

(Crick and Symonds, 1970)

Crick, M.F., and Symonds, A.J. 'Software Associative Memory for Complex Data Structures'. IBM Cambridge Scientific Center, Report No. G320-2058, 1970.

(Curtice, 1976)

Curtice, R. 'The Outlook for Data Base Management'. DATAMATION, 22, 4 (April 1976), 46-49.

(Date, 1975)

Date, C.J. An Introduction to Database Systems. Addison-Wesley, Reading, MA, 1975.

(DeFiore and Berra, 1973)

DeFiore, C.R., and Berra, P.B. 'A Data Management System Utilizing an Associative Memory'. NCC, 42 (1973), 181-185.

(Denning, 1970)

Denning, P.J. 'Virtual Memory'. ACM Computing Surveys, 2, 3 (September 1970), 153-190.

(Denning, 1971)

Denning, P.J. 'Third Generation Computing Systems'. ACM Computing Surveys, 3, 4 (December 1971), 175-215.

(Dijkstra, 1968)

Dijkstra, E.W. 'The Structure of T.H.E. Multiprogramming System'. CACM, 11, 5 (May 1968).

(Donovan, 1972)

Donovan, J.J. Systems Programming. McGraw-Hill, New York, 1972.

(Donovan, 1976)

Donvan, J.J. 'Database System Approach to Management Decision Support'. TODS, 4, 1 (December 1976), 344-369.

(Donovan et. al., 1975)

Donovan, J.J., 'An Application of a Generalized Management Information System to Energy Policy and Decision Making - The User's View'. NCC, 44, (1975).

(Donovan and Jacoby, 1975)

Donovan, J.J., and Jacoby, H.D. 'A Hierarchical Approach to Information System Design'. CISR-5, Sloan School of Management, MIT, January 1975.

(Donovan and Madnick, 1977)

Donovan, J.J. and Madnick, S.E. 'Institutional and Ad Hoc Decision Support Systems and Their Effective Use'. Data Base, 8, 2 (Winter 1977), 79-88.

(Dugan et. al., 1966)

Dugan, J.A. 'A Study of the Utility of Associative Memory Processors'. Proceedings of the 21st ACM National Conference, (1966), 347-360.

(Earley, 1971)

Earley, J. 'Toward an Understanding of Data Structures'. CACM, 14, 10 (October 1971), 617-627.

(Eckert, 1976)

Eckert, J.P. 'Thoughts on the History of Computing'. Computer, 9, 12 (December 1976), 58-65.

(Edelson, 1977)

Edelson, B. 'Satellite Communications'. Science, 195, 4283 (March 18, 1977), 1125-1133.

(Elspas, 1972)

Elspas, B. 'An Assessment of Techniques for Proving Program Correctness'. ACM Computing Surveys, 4, 2 (June 1972), 97-147.

(Farber and Baran, 1977)

Farber, D., and Baren, P. 'The Convergence of Computing and Telecommunication Systems'. Science, 195, 4283 (March 18, 1977), 1166-1169.

(Folinus, 1974)

Folinus, J.J. 'Virtual Information in Database Systems'. Working Paper CISR-3, Sloan School of Management, MIT, July 1974.

(Forrester, 1975)

Forrester, J.W. 'Dynamics of Socio-Economic Systems'. Report No. D-2230-1, Systems Dynamics Group, MIT, August 1975.

(Frankenberg, 1977)

Frankenberg, R. 'Unraveling the Mystery in User Microprogramming'. Mini-Micro System, 10, 6 (June 1977), 28-33.

(Fuller et. al., 1976)

Fuller, S.H., Lesser, V.R., Bell, G.C., and Kaman, C.H. 'The Effects of Emerging Technology and Emulation Requirements on Microprogramming'. IEEE Transactions on Computers, C-25, 10 (October 1976), 1000-1009.

(Gray et. al., 1975)

Gray, J.N., Lorie, R.A., and Putzolu, G.R. 'Granularity of Locks in a Shared Data Base'. VLDB, September 1975.

(Greenberg and Webber, 1975)

Greenberg, B.S., and Webber, S.H. 'MULTICS Multilevel Paging Hierarchy'. IEEE INTERCON, 1975.

(Hakozaki et. al., 1977)

Hakozaki, K., Mizuma, M., Kakino, T., Umemura, M., and Hiyoshi, S. 'A Conceptual Design of a Generalized Database System'. VLDB, 1977, 246-253.

(Harker and Hsu, 1972)

Harker, J.M., and Hsu, C. 'Magnetic Disks for Bulk Storage - Past and Future'. AFIP Conference Proceedings, 40 (1972), 945-955.

(Hartigan, 1975)

Hartigan, J. Clustering Algorithms. Wiley Interscience, 1975.

(Hassitt and Lyon, 1976)

Hassitt, A., and Lyon, L.E. 'An APL Emulator on System/370'. IBMSJ, 15, 4 (2976), 358-378.

(Hatfield, 1972)

Hatfield, D.J. 'Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance'. IBMJRD, 16, 1 (January 1972), 58-66.

(Haughton, 1975)

Haughton, K.E. 'An Overview of Disk Storage Systems'. Proceedings IEEE, 63, 8 (August 1975), 1148-1152.

(Heacox et. al., 1975)

Heacox, H.C., Cosloy, E.S., and Cohen, J.B. 'An Experiment in Dedicated Data Management'. VLDB, September 1975,511-513.

(Healy et. al., 1972)

Healy, L.D., Doty, K.L., and Lipovski, G.J. 'The Architecture of a Context Addressed Segment Sequential Storage'. AFIPS Conference Proceedings, 41, (1972), 691-701.

(Held et. al., 1975)

Held, G.D., Stonebraker, M., and Wong, E. 'INGRESS - A Relational Data Base Management System'. NCC, 44, (1975).

(Hodges, 1975)

Hodges, D.A. 'A Review and Projection of Semiconductor Components for Digital Storage'. Proceedings IEEE, 63, 8 (August 1975), 1136-1147.

(Hollaar, 1974)

Hollaar, L.A. 'A List Merging Processor for Information Retrieval Systems'. Proceedings of the Workshop on Architecture for Non-numeric Processing, Dallas, Texas, October 1974.

(Hollomon, 1975)

Hollomon, H. 'Produce More to Stay Ahead'. Technology Review, 77, (January 1975), 57.

(Horning and Randell, 1973)

Horning, J.J., and Randell, B. 'Process Structuring'. ACM Computing Surveys, 5, 1, (March 1973), 5-30.

(Hsiao, 1975)

Hsiao, D.K. The Database Systems. Addison-Wesley, 1975.

(Hsiao and Harary, 1970)

Hsiao, D.K., and Harary, F. 'A Formal System for Information Retrieval from Files'. CACM, 13, 2 (February 1970).

(Hsiao and Kannan, 1976)

Hsiao, D.K., and Kannan, K. 'The Architecture of a Database Computer - Part II: The Design of Structure Memory and its Related Processors'. OSU-CISRC-TR-76-e, Ohio State University, December 1976.

(Huff and Madnick, 1978)

Huff, S.L., and Madnick, S.E., 'An Extended Model for a Systematic Approach to the Design of Complex Systems. MIT CISR Internal Report No. P010-7806-07, July 1978.

(Hughes Aircraft Company, 1973)

Hughes Aircraft Company. 'MTACCS Test Bed System Description'. Report prepared for US Naval Electronics Systems Command, Contract N00039-70-c, June 1973.

(Hughes et. al., 1975)

Hughes, W.C. et. al. 'A Semiconductor Nonvolatile Electron-Beam Accessed Mass Memory'. Proceedings IEEE, 63, 8 (August 1975), 1230-1240.

(Irwin and Johnson, 1977)

Irwin, M., and Johnson, S. 'The Information Economy and Public Policy'. Science, 195, 4283 (March 18, 1977), 1170-1174.

(Johnson, 1975)

Johnson, C. 'IBM 3850 - Mass Storage System'. AFIPS Conference Proceedings, 44, (1975), 509-514.

(Kannan and Hsiao, 1977)

Kannan, K. and Hsiao, D.K. 'The Role of Emerging Technologies in Building Large On-line Database Systems'. Proceedings of the IEEE Workshop on Picture Data Definition and Management, 1977.

(Kluge, 1978)

Kluge, W.E. 'Data File Management in Shift-Register Memories' TODS, 3, 2(June 1978), 159-177.

(Lang et. al., 1977)

Lang, T., Nahouraii, E. Kasuga, K. and Fernadez, E.B. 'An Architectural Extension for a Large Database System Incorporating a Processor for Disk Search'. VLDB, 1977, 204-210.

(Langdon, 78)

Langdon G.G. Jr. 'A Note on Associative Processing for Data Management'. TODS, 3, 2 (June 1978), 148-158.

(Lin et. al., 1976)

Lin, S.C., Smith, D.C.P., and Smith, J.M. 'The Design of a Rotating Associative Memory for Relational Database Applications'. TODS, 1, 1 (March 1976), 53-75.

(Line et. al., 1973)

Line, R.R., Gates, R., and Peng, T. 'Associative Processor Applications to Real Time Data Management'. NCC, 42, (1973), 187-195.

(Lorie, 1974)

Lorie, R.A. 'XRM - An Extended (N-ary) Relational Memory'. Report No. G320-2096, IBM Cambridge Scientific Center, January 1974.

(Lum, 1971)

Lum, V.Y., et. al. 'Key-to-address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files'. CACM, 14, 4, (April 1971), 228-239.

(Madnick, 1970)

Madnick, S.E. 'Design Strategies for File Systems'. MIT Project MAC Report No. TR-78, October 1970.

(Madnick, 1973a)

Madnick, S.E. 'Storage Hierarchy Systems'. MIT Project MAC Report No. TR-105, 1973.

(Madnick, 1973b)

Madnick, S.E. 'The Future of Computers'. Technology Review, 75, 8 (July/August 1973), 34-45.

(Madnick, 1975a)

Madnick, S.E. 'Design of a General Hierarchical Storage System'. IEEE INTERCON Proceedings, 1975, 1-7.

(Madnick, 1975b)

Madnick, S.E. 'INFOPLEX - Hierarchical Decomposition of a Large Information Management System Using a Microprocessor Complex'. NCC, 44, (May 1975), 581-587.

(Madnick, 1977)

Madnick, S.E. 'Trends in Computers and Computing: The Information Utility'. Science, 195, 4283 (1973), 1191-1199.

(Madnick and Alsop, 1969)

Madnick, S.E., and Alsop, J. 'A Modular Approach to File System Design'. SJCC Proceedings, 34 (May 1969), 1-14.

(Madnick and Donovan, 1974)

Madnick, S.E., and Donovan, J.J. 'Operating Systems'. McGraw-Hill, New York, 1974.

(Madnick and Goldberg, 1976)

Madnick, S.E., and Goldberg, R.P. 'Concepts and Facilities' Report FOS-22, Report prepared for the US Navy Electronics Laboratory Center, San Diego, California.

(Marill and Stern, 1975)

Marill, T., and Stern, D. 'The Datacomputer - a Network Data Utility'. AFIPS Conference Proceedings, 44 (975), 389-395.

(Martin, 1975)

Martin, J. Computer Data Base Organization. Prentice-Hall, New York, 1975.

(Martin and Frankel, 1975)

Martin, R.R., and Frankel, H.D. 'Electronic Disks in the 1980's'. Computer, 8, 2 (February 1975), 24-30.

(Mattson et. al., 1970)

Mattson, R., et. al. 'Evaluation Techniques for Storage Hierarchies'. IBMSJ, 9, 2 (1970), 78-117.

(McCabe, 1978)

McCabe, E.J. 'Locality in Logical Database Systems: A Framework for Analysis', MIT Sloan School MS Thesis, 1978.

(Meade, 1970)

Meade, R.M. 'On Memory System Design'. AFIPS Conference Proceedings, 37, (1970), 33-34.

(Mesarovic et. al., 1970)

Mesarovic, M.D. et. al. Theory of Multilevel Hierarchical Systems. Academic Press, New York, 1970.

(Miller, 1977)

Miller, S.E. 'Photons in Fibers for Telecommunication'. Science, 195, 4283 (March 18, 1977), 1211-1215.

(Mills, 1977)

Mills, H.D. 'Software Engineering'. Science, 195, 4283 (March 18, 1977), 1199-1204.

(Minsky, 1972)

Minsky, N. 'Rotating Storage Devices as Partially Associative Memories'. NCC, 41 (1972), 587-596.

(Moulder, 1973)

Moulder, R. 'An Implementation of a Data Management on an Associative Processor'. AFIPS Conference Proceedings, 42, (1973), 171-176.

(Mueller, 1976)

Mueller, G. Computer, 9, 12 (December 1976), 100.

(Mulvany, 1974)

Mulvany, R.B. 'Engineering Design of a Disk Storage Facility with Data Modules'. IBMJRD, 18, 6, (November 1974).

(Myers, 1976)

Myers, W. 'Key Developments in Computing Technology: A Survey'. Computer, 9, 11 (Novemebr 1976), 48-77.

(Nolan, 1973)

Nolan, R.L. 'Computer Data Bases - The Future is Now'. Harvard Business Review, 51, 5, (September/October 1973), 98-114.

(Noyce, 1977)

Noyce, R. 'Large Scale Integration - What is Still to Come?'. Science, 195, 4283 (March 18, 1977), 1120-1106.

(Ohnigian, 1975)

Ohnigian, S. 'Random File Processing in a Storage Hierarchy'. IEEE INTERCON, 1975.

(Ornstein et. al., 1975)

Ornstein, S.M. et. al. 'PLURIBUS - A Reliable Multiprocessor'. AFIPS Conference Proceedings, 44, (1975), 551-560.

(Ozkarahan et. al., 1977)

Ozkarahan, E.A., Schuster, S.A., and Sevcik, K.C.

'Performance Evaluation of a Relational Associative Processor'. TODS, 2, 2 (June 1977), 175-195.

(Ozkarahan, 1975)

Ozkarahan, E.A., Schuster, S.A., and Smith, K.C. 'RAP - Associative Processor for Data Base Management'. AFIPS Conference Proceedings, 44, (1975), 379-388.

(Palmer, 1967)

Palmer, I.R. 'Levels of Database Description'. IFIP Proccedings, 1967.

(Palmer, 1975)

Palmer, I.R. 'Data Base Systems: A Practical Reference'. Q.E.D. Information Science Ltd, Wellesley, MA, 1975.

(Panigrahi, 1976)

Panigrahi, G. 'Charge-Coupled Memories for Computer Systems'. Computer, 9, 4 (April 1976), 33-41.

(Parhami, 1972)

Parhami, B. 'A Highly Parallel Computer System for Information Retrieval'. AFIPS Conference Proceedings, 41, (1972), 681-690.

(Parhami, 1973)

Parhami, B. 'Associative Memories and Processors: An Overview and Selected Bibliography'. Proceedings of the IEEE, 61, 6 (June 1973), 722-730.

(Parker, 1971)

Parker, J.L. 'A Logic Per Track Retrieval System'. Proceedings IFIP Congress 1971, North-Holland Publishing Co., 1971.

(Parnas, 1976)

PArnas, D.L. 'On The Design and Development of Program Families'. IEEE Transactions on Software Engineering, SE-2-1, March 1976.

(Pattee, 1973)

Pattee, H.H. Hierarch Theory: The Challenge of Complex Systems. George Brazillier, New York, 1973.

(Potter, 1977)

Potter, R.J. 'Electronic Mail'. Science, 195, 4283 (March 18, 1977), 1223-1229.

(Ramamoothy and Chandy, 1970)

Ramamoorthy, C.V., and Chandy, K.M. 'Optimization of Memory Hierarchies in Multiprogrammed Systems'. Journal of the ACM, July 1970.

(Rothnie, 1975)

Rothnie, J.B. 'Evaluating Inter-entry Retrieval Expressions in a Relational Database Management System'. AFIPS Conference Proceedings, 1975.

(Sarmento and Chen, 1977)

Sarmento, L.E.S., and Chen, P.P. 'Performance Evaluation of Generalized Management Information System on a Virtual Memory Operating and Computer System - A Case Study of GMIS on VM/370'. Workshop on Operating and Database Management Systems, Northwestern University, March 21-22, 1977.

(Schmidt, 1974)

Schmidt, D. 'Monolthic Processors'. Computer Design, October 1974, 88-95.

(Schuster, et al., 1976)

Schuster, S.A., Ozkaraham, E.A., and Smith, K.C. 'A Virtual Memory System for a Relational Associative Processor'. NCC, 1976, 855-862.

(Severance, 1974)

Severance, D.G. 'A Parametric Model of Alternative File Structures'. Information Systems, 1, 2 (June 1974), 51-55.

(Senko, 1976)

Senko, M.E. 'DIAM II: The Binary Infological Level and its Database Language - FORAL'. Proceedings of the ACM Conference on Data. March 1976, 1-21.

(Sherf, 1974)

Sherf, J. 'Data Security - A Comprehensive and Annotated Bibliography'. MIT PRoject MAc Report No. TR-122, January, 1974)

(Slotnick, 1970)

Slotnick, D.L. 'Logic Per Track Devices'. Advances in

Computers, 10, Franz Alt, Ed., Academic Press, New York, 1970, 291-296.

(Smith and Chang, 1975)

Smith, J.M., and Chang, P.Y. 'Optimizing the Performance of a Relational Algebra Data Base Interface'. CACM, 18, 10 (October 1975), 568-579.

(Su, 1977)

Su, S.Y.W. 'Associative Programming in CASSM and its Applications'. VLDB, 1977, 213-228.

(Su and Lipovski, 1975)

Su, S.Y., and Lipovski, G.J. 'CASSM: A Cellular System for Very Large Databases'. VLDB, September 1975, 456-472.

(Toh et. al., 1977)

Toh, T., Kawazu, S., and Suzuki, K. 'Multi-Level Structures of the DBTG Data Model for an Achievement of the Physical Independence'. VLDB, 1977, 403-414.

(Toong, 1975)

Toong, H.D. 'Microprocessor-based multiprocessor ring structured network'. NCC, 1975.

(Tweedt, 1972)

Tweedt, K. 'World-wide Data Mangement Systems for WWMCCS'. Proceedings DoD Joint ADP Conference, III, October 1972, 8.1-8.22.

(Verity, 1976)

Verity, J. 'Software Independent Optimistic as Data Processing Budgets Increases'. Electronics News, January 3, 1977, 28.

(Weissberger, 1977)

Weissberger, A.J. 'Analysis of Multiple Microprocessor System Architectures'. Computer Design, June 1977, 7.46-7.58.

(Wensley, 1975)

Wensley, J.H. 'The Impact of Electronic Disks on System Architecture'. Computer, 8, 2 (February), 24-30.

(Withington, 1975)

Withington, F.G. 'Beyond 1984: A Technology Forecast'. Datamation, 21, 1 (January 1975), 54-73.

(Yeh et. al., 1977)

Yeh, R.T., and Baker, J.W. 'Toward a Design Methodology for DBMS: A Software Engineering Approach'. VLDB, 1977, 16-27.

(Zloof, 1975)

Zloof, M.M. 'Query By Example'. NCC, 44, (May 1975), 431-438.